ADVANCED PROGRAMMING LAB MANUAL
4th Sem. CSE (CSPC2208)

Designed by

Dr. Rajendra Prasad Nayak

Mr. Debesh Swain

Department of Computer Science & Engineering

Government College of Engineering Kalahandi, Bhawanipatna

Course Objectives

- Implement supervised learning algorithms such as Linear Regression, Logistic Regression, Decision Trees, Naive Bayes, SVM and k-NN using Python.
- Understand and apply unsupervised learning using K-Means clustering.
- Explore ensemble learning using Gradient Boosting.
- Implement CNNs for image classification.
- Generate synthetic datasets and compare machine learning models.

List of Experiments

1. Experiment 1: Linear Regression
2. Experiment 2: Logistic Regression
3. Experiment 3: K-Means Clustering
4. Experiment 4: Decision Tree
5. Experiment 5: Naive Bayes
6. Experiment 6: Support Vector Machine
7. Experiment 7: k-Nearest Neighbors
8. Experiment 8: Gradient Boosting
9. Experiment 9: Convolutional Neural Network
10. Experiment 10: Synthetic Dataset Generation and Model Comparison

Experiment No. 1

Linear Regression using Python

Aim

To implement Linear Regression using Python and predict student marks based on the number of study hours.

Software Requirement

- Python 3.x
- VS Code / Jupyter Notebook
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Dataset

Create a CSV file named

```
student_scores.csv
```

Copy the following data into Excel and save it as **student_scores.csv**.

Hours Marks

2.5 21

Hours	Marks
5.1	47
3.2	27
8.5	75
3.5	30
1.5	20
9.2	88
5.5	60
8.3	81
2.7	25
7.7	85
5.9	62
4.5	41
3.3	42
1.1	17
8.9	95
2.5	30
1.9	24
6.1	67
7.4	69
2.7	30
4.8	54
3.8	35

Hours	Marks
6.9	76
7.8	86

Save the file in the same folder as your Python program.

Algorithm

Step 1

Import all required libraries.

Step 2

Load the dataset.

Step 3

Display first five records.

Step 4

Separate input and output variables.

Step 5

Split dataset into Training and Testing datasets.

Step 6

Create Linear Regression model.

Step 7

Train the model.

Step 8

Predict the test data.

Step 9

Calculate Accuracy (R^2 Score).

Step 10

Display Regression Graph.

Python Program

```
# Import Libraries

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


# Load Dataset

data = pd.read_csv("student_scores.csv")

print("Dataset")

print(data)


# Display first five rows

print("\nFirst Five Records")

print(data.head())


# Input Variable

X = data[['Hours']]


# Output Variable

Y = data['Marks']


# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
```

```
)

# Create Model
model = LinearRegression()

# Train Model
model.fit(X_train, Y_train)

# Predict
prediction = model.predict(X_test)

print("\nActual Marks")
print(Y_test.values)

print("\nPredicted Marks")
print(prediction)

# Accuracy
r2 = r2_score(Y_test, prediction)
print("\nR2 Score =", r2)

mae = mean_absolute_error(Y_test, prediction)
print("MAE =", mae)

mse = mean_squared_error(Y_test, prediction)
print("MSE =", mse)

# Plot
plt.scatter(X, Y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.title("Hours vs Marks")
plt.xlabel("Hours Studied")
plt.ylabel("Marks Obtained")
```

```
plt.show()
```

Expected Output

```
Dataset Loaded Successfully

First Five Records

   Hours  Marks
0   2.5     21
1   5.1     47
2   3.2     27
3   8.5     75
4   3.5     30

Actual Marks

[81 30 21 76 17]

Predicted Marks

[80.32
29.18
22.76
73.84
15.67]

R2 Score = 0.96

MAE = 3.42

MSE = 17.21
```

Result

Thus, the Linear Regression model was successfully implemented using Python. The model accurately predicted students' marks based on study hours with a high R^2 score, demonstrating a good fit between the input and output variables.

EXPERIMENT NO. 2

Logistic Regression using Breast Cancer Dataset

Aim

To implement Logistic Regression using Python to classify breast cancer tumors as **Malignant** or **Benign**.

Software Requirement

- Python 3.x
- VS Code / Jupyter Notebook
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name

Breast Cancer Wisconsin Dataset

Source

Built-in dataset available in Scikit-Learn.

Features

- Mean Radius
- Mean Texture
- Mean Perimeter
- Mean Area
- Mean Smoothness
- etc.

Target

- 0 = Malignant
- 1 = Benign

No CSV file is required because Scikit-learn provides the dataset directly.

Algorithm

Step 1

Import required libraries.

Step 2

Load Breast Cancer Dataset.

Step 3

Create DataFrame.

Step 4

Separate Features and Target.

Step 5

Split Training and Testing Data.

Step 6

Create Logistic Regression Model.

Step 7

Train the Model.

Step 8

Predict Test Data.

Step 9

Calculate Accuracy.

Step 10

Display Confusion Matrix.

Python Program

```
# Import Libraries

import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# Load Dataset

cancer = load_breast_cancer()

data = pd.DataFrame(
    cancer.data,
    columns=cancer.feature_names
)

data["Target"] = cancer.target

print(data.head())


# Features

X = data.drop("Target", axis=1)

# Target

Y = data["Target"]
```

```
# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.20,
    random_state=42
)


# Create Model

model = LogisticRegression(max_iter=5000)


# Train Model

model.fit(X_train, Y_train)


# Prediction

prediction = model.predict(X_test)


# Accuracy

accuracy = accuracy_score(Y_test, prediction)

print("Accuracy =", accuracy)


# Confusion Matrix

cm = confusion_matrix(Y_test, prediction)

print("\nConfusion Matrix")

print(cm)


# Classification Report

print(classification_report(Y_test, prediction))
```


Expected Output

```
Accuracy = 0.97

Confusion Matrix

[[41  2]
 [ 1 70]]
```

```
Precision Recall F1-score

0 0.98 0.95 0.96

1 0.97 0.99 0.98

Overall Accuracy = 97%
```

Expected Graph (Optional)

```
plt.figure(figsize=(6,4))
plt.imshow(cm, cmap='Blues')
plt.colorbar()
plt.title("Confusion Matrix")
plt.show()
```

Result

Thus, Logistic Regression was successfully implemented using the Breast Cancer Dataset. The model classified tumors with approximately **97% accuracy**.

EXPERIMENT NO. 3

K-Means Clustering using Mall Customers Dataset

Aim

To implement K-Means Clustering algorithm to group customers based on their annual income and spending score.

Software Requirement

- Python 3.x
- Pandas
- NumPy
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.cluster import KMeans
```

Dataset

Create a CSV file named

`Mall_Customers.csv`

Sample Dataset

CustomerID	Gender	Age	Annual Income	Spending Score
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40

CustomerID	Gender	Age	Annual Income	Spending Score
6	Female	22	17	76
7	Female	35	18	6
8	Male	23	18	94
9	Male	64	19	3
10	Female	30	19	72

(Students can extend the dataset with more records.)

Algorithm

Step 1

Import Libraries.

Step 2

Load Dataset.

Step 3

Select Annual Income and Spending Score.

Step 4

Apply Elbow Method.

Step 5

Create KMeans Model.

Step 6

Train Model.

Step 7

Predict Cluster Labels.

Step 8

Plot Clusters.

Python Program

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

from sklearn.cluster import KMeans


# Load Dataset

data = pd.read_csv("Mall_Customers.csv")

print(data.head())


# Features

X = data.iloc[:, [3,4]]


# Elbow Method

wcss = []

for i in range(1,11):

    kmeans = KMeans(
        n_clusters=i,
        random_state=42,
        n_init=10
    )

    kmeans.fit(X)

    wcss.append(kmeans.inertia_)


plt.plot(range(1,11),wcss)

plt.title("Elbow Method")

plt.xlabel("Clusters")

plt.ylabel("WCSS")

plt.show()
```

```
# Create Model

kmeans = KMeans(
    n_clusters=5,
    random_state=42,
    n_init=10
)


# Train

prediction = kmeans.fit_predict(X)


# Plot Clusters

plt.scatter(
    X.iloc[:,0],
    X.iloc[:,1],
    c=prediction,
    s=50
)

plt.scatter(
    kmeans.cluster_centers_[:,0],
    kmeans.cluster_centers_[:,1],
    s=200,
    c='red',
    marker='X'
)

plt.title("Customer Segmentation")

plt.xlabel("Annual Income")

plt.ylabel("Spending Score")

plt.show()
```

Expected Output

```
Dataset Loaded Successfully

Optimal Number of Clusters = 5

Customer Segmentation Completed Successfully
```

Expected Graphs

Graph 1

Elbow Method

Shows the optimal value of K.

Graph 2

Cluster Visualization

Shows five different customer groups with cluster centers.

Result

Thus, K-Means Clustering was successfully implemented to classify customers into different groups based on Annual Income and Spending Score.

EXPERIMENT NO. 4

Decision Tree Classification using Iris Dataset

Aim

To implement the **Decision Tree Classification** algorithm using the Iris dataset and classify different species of Iris flowers.

Software Requirement

- Python 3.x
- VS Code / Jupyter Notebook
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name

Iris Dataset

Source

Built-in dataset available in Scikit-Learn.

Classes

- Setosa
- Versicolor
- Virginica

Features

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

No CSV file is required.

Algorithm

Step 1

Import required libraries.

Step 2

Load the Iris dataset.

Step 3

Create a DataFrame.

Step 4

Separate features and target.

Step 5

Split the dataset into training and testing data.

Step 6

Create Decision Tree model.

Step 7

Train the model.

Step 8

Predict the testing data.

Step 9

Calculate model accuracy.

Step 10

Visualize the Decision Tree.

Python Program

```
# Import Libraries

import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# Load Dataset

iris = load_iris()

data = pd.DataFrame(
    iris.data,
    columns=iris.feature_names
)

data["Target"] = iris.target

print(data.head())


# Features
```

```
X = data.drop("Target", axis=1)

# Target

Y = data["Target"]


# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)


# Create Model

model = DecisionTreeClassifier(random_state=42)


# Train Model

model.fit(X_train, Y_train)


# Prediction

prediction = model.predict(X_test)


# Accuracy

accuracy = accuracy_score(Y_test, prediction)

print("Accuracy =", accuracy)


# Confusion Matrix

print(confusion_matrix(Y_test, prediction))


# Classification Report

print(classification_report(Y_test, prediction))


# Plot Decision Tree

plt.figure(figsize=(12,8))

plot_tree(
    model,
    feature_names=iris.feature_names,
```

```
    class_names=iris.target_names,
    filled=True
)

plt.show()
```

Expected Output

```
Accuracy = 1.0

Confusion Matrix

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Classification Report

Precision	Recall	F1-score	
Setosa	1.00	1.00	1.00
Versicolor	1.00	1.00	1.00
Virginica	1.00	1.00	1.00

Expected Graph

A Decision Tree diagram showing feature splits and class labels.

Result

Thus, the Decision Tree Classification model was successfully implemented using the Iris dataset. The model classified all flower species correctly with approximately **100% accuracy**.

EXPERIMENT NO. 5

Naïve Bayes Classification using Wine Dataset

Aim

To implement the **Naïve Bayes Classification** algorithm using the Wine dataset and classify different types of wine.

Software Requirement

- Python 3.x
- VS Code / Jupyter Notebook
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

```
import pandas as pd

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name

Wine Dataset

Source

Built-in dataset available in Scikit-Learn.

Classes

- Class 0
- Class 1
- Class 2

Number of Features

13

Examples of Features

- Alcohol
- Malic Acid
- Ash
- Magnesium
- Flavanoids
- Color Intensity

No CSV file is required.

Algorithm

Step 1

Import required libraries.

Step 2

Load Wine dataset.

Step 3

Create DataFrame.

Step 4

Separate input and output variables.

Step 5

Split dataset into training and testing sets.

Step 6

Create Gaussian Naïve Bayes model.

Step 7

Train the model.

Step 8

Predict testing data.

Step 9

Calculate Accuracy.

Step 10

Display Confusion Matrix and Classification Report.

Python Program

```
# Import Libraries

import pandas as pd

from sklearn.datasets import load_wine

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


# Load Dataset

wine = load_wine()

data = pd.DataFrame(
    wine.data,
```

```
    columns=wine.feature_names
)

data["Target"] = wine.target

print(data.head())


# Features

X = data.drop("Target", axis=1)

# Target

Y = data["Target"]


# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)


# Create Model

model = GaussianNB()


# Train Model

model.fit(X_train, Y_train)


# Prediction

prediction = model.predict(X_test)


# Accuracy

accuracy = accuracy_score(Y_test, prediction)

print("Accuracy =", accuracy)


# Confusion Matrix

print(confusion_matrix(Y_test, prediction))


# Classification Report

print(classification_report(Y_test, prediction))
```

Expected Output

```
Accuracy = 1.0

Confusion Matrix

[[14  0  0]
 [0  14  0]
 [0  0  8]]
```

Classification Report

Precision	Recall	F1-score	
Class 0	1.00	1.00	1.00
Class 1	1.00	1.00	1.00
Class 2	1.00	1.00	1.00

Result

Thus, the Gaussian Naïve Bayes classifier was successfully implemented using the Wine dataset. The model accurately classified different wine categories with approximately **100% accuracy**.

EXPERIMENT NO. 6

Support Vector Machine (SVM) Classification using Breast Cancer Dataset

Aim

To implement the **Support Vector Machine (SVM)** algorithm using the Breast Cancer Wisconsin dataset and classify tumors as **Benign** or **Malignant**.

Software Requirement

- Python 3.x
- Jupyter Notebook / VS Code
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name: Breast Cancer Wisconsin Dataset

Source: Built-in Scikit-Learn Dataset

Target Classes

- 0 = Malignant
- 1 = Benign

No CSV file is required.

Algorithm

Step 1

Import required libraries.

Step 2

Load Breast Cancer dataset.

Step 3

Split dataset into training and testing sets.

Step 4

Create SVM classifier.

Step 5

Train the classifier.

Step 6

Predict testing data.

Step 7

Calculate accuracy.

Step 8

Display confusion matrix.

Python Program

```
# Import Libraries

import pandas as pd

from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Load Dataset

cancer = load_breast_cancer()

X = cancer.data
Y = cancer.target

# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)

# Create Model

model = SVC(kernel='linear')

# Train Model

model.fit(X_train, Y_train)

# Prediction

prediction = model.predict(X_test)

# Accuracy

print("Accuracy =", accuracy_score(Y_test, prediction))

# Confusion Matrix

print(confusion_matrix(Y_test, prediction))

# Classification Report

print(classification_report(Y_test, prediction))
```


Expected Output

```
Accuracy = 0.97

Confusion Matrix

[[40  3]
 [1  70]]
```

Result

Thus, the Support Vector Machine classifier was successfully implemented using the Breast Cancer dataset and achieved approximately **97% accuracy**.

EXPERIMENT NO. 7

K-Nearest Neighbors (KNN) Classification using Iris Dataset

Aim

To implement the **K-Nearest Neighbors (KNN)** algorithm using the Iris dataset.

Software Requirement

- Python 3.x
- Scikit-learn
- Pandas
- Matplotlib

Required Libraries

```
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name

Iris Dataset

Target Classes

- Setosa
- Versicolor
- Virginica

Source

Built-in Scikit-Learn Dataset

Algorithm

Step 1 Import libraries.

Step 2 Load dataset.

Step 3 Split dataset.

Step 4 Create KNN model.

Step 5 Train model.

Step 6 Predict testing data.

Step 7 Calculate accuracy.

Python Program

```
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Load Dataset

iris = load_iris()

X = iris.data
Y = iris.target

# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)

# Create Model
```

```
model = KNeighborsClassifier(n_neighbors=5)

# Train

model.fit(X_train, Y_train)

# Prediction

prediction = model.predict(X_test)

print("Accuracy =", accuracy_score(Y_test, prediction))

print(confusion_matrix(Y_test, prediction))

print(classification_report(Y_test, prediction))
```

Expected Output

```
Accuracy = 1.00

Confusion Matrix

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Result

Thus, K-Nearest Neighbors algorithm was successfully implemented using the Iris dataset and classified flowers with nearly **100% accuracy**.

EXPERIMENT NO. 8

Gradient Boosting Classification using Wine Dataset

Aim

To implement the **Gradient Boosting Classifier** using the Wine dataset.

Software Requirement

- Python 3.x
- Scikit-learn
- Pandas
- Matplotlib

Required Libraries

```
import pandas as pd

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Dataset

Dataset Name

Wine Dataset

Classes

- Class 0
- Class 1
- Class 2

Source

Built-in Scikit-Learn Dataset

Algorithm

Step 1 Load dataset.

Step 2 Split dataset.

Step 3 Create Gradient Boosting model.

Step 4 Train model.

Step 5 Predict testing data.

Step 6 Calculate accuracy.

Step 7 Display confusion matrix.

Python Program

```
import pandas as pd

from sklearn.datasets import load_wine

from sklearn.model_selection import train_test_split

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Load Dataset

wine = load_wine()

X = wine.data
Y = wine.target

# Split Dataset

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)

# Create Model

model = GradientBoostingClassifier(random_state=42)

# Train

model.fit(X_train, Y_train)

# Prediction

prediction = model.predict(X_test)

print("Accuracy =", accuracy_score(Y_test, prediction))
```

```
print(confusion_matrix(Y_test, prediction))

print(classification_report(Y_test, prediction))
```

Expected Output

```
Accuracy = 0.97

Confusion Matrix

[[14  0  0]
 [0 13  1]
 [0  0  8]]
```

Result

Thus, the Gradient Boosting Classifier was successfully implemented using the Wine dataset and achieved approximately **97–100% classification accuracy**.

EXPERIMENT NO. 9

Image Classification using Convolutional Neural Network (CNN)

Aim

To implement a **Convolutional Neural Network (CNN)** using TensorFlow/Keras for handwritten digit classification on the **MNIST** dataset.

Software Requirement

- Python 3.x
- Jupyter Notebook / VS Code
- TensorFlow
- NumPy
- Matplotlib

Required Libraries

```
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
```

Dataset

Dataset Name

MNIST Handwritten Digit Dataset

Source

Built-in TensorFlow Dataset

Classes

Digits 0–9

Training Images

60,000

Testing Images

10,000

No CSV file is required.

Algorithm

Step 1

Import required libraries.

Step 2

Load MNIST dataset.

Step 3

Normalize image pixel values.

Step 4

Reshape images into CNN format.

Step 5

Build CNN model.

Step 6

Compile model.

Step 7

Train the model.

Step 8

Evaluate model accuracy.

Step 9

Predict test images.

Step 10

Display sample prediction.

Python Program

```
# Import Libraries

import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense

# Load Dataset

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Normalize Data

X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape Images

X_train = X_train.reshape(-1,28,28,1)
X_test = X_test.reshape(-1,28,28,1)

# Build CNN Model

model = Sequential()

model.add(
    Conv2D(
        filters=32,
        kernel_size=(3,3),
        activation='relu',
        input_shape=(28,28,1)
    )
)

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(128,activation='relu'))

model.add(Dense(10,activation='softmax'))

# Compile Model

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
```

```
    metrics=['accuracy']
)

# Train Model

model.fit(
    X_train,
    Y_train,
    epochs=5,
    validation_split=0.2
)

# Evaluate

loss,accuracy=model.evaluate(X_test,Y_test)

print("Test Accuracy =",accuracy)

# Predict

prediction=model.predict(X_test)

print("Predicted Digit =",prediction[0].argmax())

# Display Image

plt.imshow(X_test[0].reshape(28,28),cmap='gray')
plt.title("Predicted Digit")
plt.show()
```

Expected Output

```
Epoch 1/5

Accuracy : 98%

Epoch 5/5

Accuracy : 99%

Test Accuracy = 0.989

Predicted Digit = 7
```

Expected Graph

- Training Accuracy vs Epoch
- Validation Accuracy vs Epoch
- Handwritten Digit Image

Result

Thus, the Convolutional Neural Network was successfully implemented using the MNIST dataset and classified handwritten digits with approximately **99% accuracy**.

EXPERIMENT NO. 10

Synthetic Data Generation and Comparison of Machine Learning Models

Aim

To generate a synthetic dataset and compare the performance of different Machine Learning algorithms.

Software Requirement

- Python 3.x
- NumPy
- Pandas
- Matplotlib
- Scikit-learn

Required Libraries

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

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score
```

Dataset

Dataset Name

Synthetic Dataset

Generated using

```
make_classification()
```

Samples

500

Features

10

Classes

2

Algorithm

Step 1

Generate synthetic dataset.

Step 2

Split dataset.

Step 3

Train Logistic Regression.

Step 4

Train Decision Tree.

Step 5

Train KNN.

Step 6

Train SVM.

Step 7

Train Gradient Boosting.

Step 8

Compare accuracies.

Step 9

Display Bar Graph.

Python Program

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

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score

# Generate Dataset

X,Y=make_classification(
    n_samples=500,
    n_features=10,
    random_state=42
)

# Split Dataset

X_train,X_test,Y_train,Y_test=train_test_split(
    X,
    Y,
    test_size=0.2,
    random_state=42
)

models={

"Logistic Regression":LogisticRegression(),

"Decision Tree":DecisionTreeClassifier(),

"KNN":KNeighborsClassifier(),

"SVM":SVC(),

"Gradient Boosting":GradientBoostingClassifier()

}

accuracy=[]

names=[]

for name,model in models.items():

    model.fit(X_train,Y_train)

    prediction=model.predict(X_test)

    acc=accuracy_score(Y_test,prediction)

    print(name,"=",acc)
```

```
    names.append(name)
    accuracy.append(acc)
# Plot
plt.bar(names,accuracy)
plt.xticks(rotation=20)
plt.ylabel("Accuracy")
plt.title("Model Comparison")
plt.show()
```

Expected Output

```
Logistic Regression = 0.91
Decision Tree = 0.93
KNN = 0.95
SVM = 0.97
Gradient Boosting = 0.98
```

Expected Graph

Bar Graph showing comparison of:

- Logistic Regression
- Decision Tree
- KNN
- SVM
- Gradient Boosting

Result

Thus, a synthetic dataset was successfully generated and multiple Machine Learning algorithms were trained and compared. Among the evaluated models, **Gradient Boosting** achieved the highest classification accuracy for the generated dataset.