Machine Learning Lab notes

Linear Regression

Aim

To implement Linear Regression on Age, Height, and Weight features to predict the Body Mass Index (BMI) of individuals.

Procedure

1. Load the dataset containing Age, Height, Weight, and BMI values.
2. Perform basic exploration and descriptive statistics.
3. Define predictor variables (Age, Height, Weight) and target variable (BMI).
4. Split the dataset into training and testing subsets.
5. Train a Linear Regression model using training data.
6. Predict BMI values for the test set.
7. Evaluate the model using Mean Squared Error (MSE) and R^2 Score.
8. Display regression coefficients and make predictions for new input values.
9. Visualize actual vs. predicted BMI values using a scatter plot.

Program

```
# Linear Regression on Age, Height, Weight, BMI Dataset

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_squared_error, r2_score

# Step 1: Load dataset

data = pd.read_csv(r"C:/Users/Karthikeyan/Downloads/bmi.csv")

# Step 2: Explore

print(data.head())

print(data.info())

print(data.describe())
```

```
# Step 3: Features and Target
X = data[["Age", "Height", "Weight"]]
y = data["Bmi"]
# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 5: Train model
model = LinearRegression()
model.fit(X_train, y_train)
# Step 6: Predictions
y_pred = model.predict(X_test)
# Step 7: Evaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
# Step 8: Coefficients
coeff_df = pd.DataFrame(model.coef_, X.columns, columns=["Coefficient"])
print(coeff_df)
# Step 9: Predict on new values
new_person = [[25, 170, 70]]
predicted_bmi = model.predict(new_person)
print("Predicted BMI:", predicted_bmi[0])
# Step 10: Visualization
plt.scatter(y_test, y_pred, color="blue")
plt.xlabel("Actual BMI")
plt.ylabel("Predicted BMI")
plt.title("Actual vs Predicted BMI")
plt.show()
```

Interpretation

The model was trained to predict BMI using Age, Height, and Weight. The evaluation results show a low error and a high R^2 value, confirming that the regression model fits the dataset effectively.

Result

Linear Regression was successfully implemented on the Age, Height, and Weight dataset to predict BMI. The model achieved good accuracy, and predictions closely matched the actual BMI values.

Random forest

Aim

To build and evaluate a machine learning model using the Random Forest Classifier for predicting passenger survival on the Titanic dataset, and to analyze model performance using accuracy, confusion matrix, classification report, and precision-recall curve.

Procedure

1. **Import Libraries**: Load essential libraries such as pandas, seaborn, sklearn, and matplotlib.
2. **Load Dataset**: Use sns.load_dataset('titanic') to import Titanipc dataset.
3. **Data Preprocessing**:
    - Drop rows with missing target (survived) values.
    - Select relevant features: pclass, sex, age, sibsp, parch, fare.
    - Convert categorical variable sex into numeric (female = 0, male = 1).
    - Fill missing values in age with the median.
4. **Train-Test Split**: Split dataset into training (80%) and testing (20%) subsets.
5. **Model Building**: Initialize a Random Forest Classifier with parameters like n_estimators=100, max_depth=10, min_samples_split=5, max_features='sqrt'.
6. **Training**: Fit the classifier on training data.
7. **Prediction**: Predict survival on test data and also generate probability estimates for the positive class.
8. **Evaluation**:
    - Calculate accuracy, confusion matrix, and classification report.
    - Plot Precision-Recall Curve.
9. **New Data Prediction**: Provide a sample passenger record and predict survival using the trained model.

Program:

```
# import necessary library
```

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.metrics import precision_recall_curve, classification_report, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

#load the titsanic dataset

titanic_data = sns.load_dataset('titanic')

print(titanic_data.head())

#  drop rows with missing target values

titanic_data=titanic_data.dropna(subset = ['survived'])

# select relevent features and target variable

x=titanic_data[['pclass','sex','age','sibsp','parch','fare']]

y=titanic_data['survived']

# convert categorical variable 'sex' into numerical using label encoding

x.loc[:,'sex']=x['sex'].map({'female':0,'male':1})

# handling missing values in the 'age' column

x.loc[:,'age'].fillna(x['age'].median(),inplace=True)

# split the datasets

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=True)

#  create randomforest classifier

rf_classifier= RandomForestClassifier(n_estimators=100,max_depth=10,

   min_samples_split=5,

   max_features='sqrt',random_state=42)

# train the classifier

rf_classifier.fit(x_train,y_train)

# make prediction on the test set

y_pred = rf_classifier.predict(x_test)

# generate problity estimates for the positive class

y_proba =rf_classifier.predict_proba(x_test)[:, 1]
```

```
# claculate precision recall curve

precision,recall,_= precision_recall_curve(y_test,y_proba)


# plot precision _recall curve

plt.figure(figsize=(8,6))

plt.plot(recall,precision,color="purple")

plt.xlabel('recall')

plt.ylabel('precision')

plt.title('precesion_recall curve for titanic model')

plt.show()


# evaluate the model

accuracy = accuracy_score(y_test,y_pred)

confusion_mat=confusion_matrix(y_test,y_pred)

classification_rep=classification_report(y_test, y_pred)

# print the result

print("confusion matrix:")

print('confusion_matrix\n',confusion_mat)

print("\naccuracy score :",accuracy)

print("\n Classification Report :\n",classification_rep)

# new value classifitation

new_data=[[3,1,55.0,0,0,71.28]]

result=rf_classifier.predict(new_data)

print(f" result of new data :{result}")
```

Interpretation:
The Random Forest model predicts Titanic passenger survival with about 81% accuracy. It balances precision and recall well, and the precision-recall curve confirms good model performance.

Result:
The model achieved ~81% accuracy, correctly classifying most passengers, and predicted the new sample passenger as Not Survived (0).

SVM

Aim

To classify the Iris flower dataset using Support Vector Machine (SVM), evaluate its performance, and visualize the results with a confusion matrix.

Procedure

The Iris dataset containing three flower species (Setosa, Versicolor, Virginica) was taken.

The dataset was divided into training set and testing set.

The data was normalized to bring all features to a common scale.

An SVM classifier with linear kernel was trained using the training data.

The trained model was tested with the testing data to measure accuracy.

Cross-validation was carried out to check the stability of the model.

A new sample flower measurement was given, and the trained model predicted its class.

The results were analyzed using accuracy score, classification report, and confusion matrix.

The confusion matrix was plotted as a heatmap for better visualization of actual vs predicted outputs.

Program

```
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,
classification_report,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load iris dataset
iris = datasets.load_iris()
X, y = iris.data, iris.target   # X = features, y = labels (0=setosa,
1=versicolor, 2=virginica)

# Split into train & test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=49, stratify=y
)
```

```python
# Scale features (important for SVM with kernels)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM classifier
clf = SVC(kernel="linear", C=1.0, gamma="scale", random_state=42)
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# cross validation
scores = cross_val_score(clf, X, y, cv=15)
print("Cross-validation scores:", scores)
print("Mean CV accuracy:", scores.mean())

# Results
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred, target_names=iris.target_names))
train_acc = clf.score(X_train, y_train)
test_acc = clf.score(X_test, y_test)
print("Training Accuracy:", train_acc)
print("Testing Accuracy:", test_acc)

# predict with new data
new_sample = np.array([[6.1, 3.5, 5.0, 5.2]])  # looks like setosa
new_sample_scaled = scaler.transform(new_sample)
prediction = clf.predict(new_sample_scaled)
print("Predicted class:", iris.target_names[prediction][0])

# Confusion Matrix Heatmap
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=iris.target_names,
            yticklabels=iris.target_names)

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix (Actual vs Predicted)")
plt.show()
```

Interpretation

The SVM classifier achieved a mean cross-validation accuracy of 98%, which shows the model is stable. The testing accuracy is 97.7%, very close to training accuracy (99%), indicating no overfitting. The classification report shows excellent precision, recall, and F1-score for all three

Iris classes, meaning the model predicts each species correctly with very few mistakes. The new flower sample was predicted as Virginica, proving the model works well for unseen data also.

Result

The SVM classifier was successfully applied on the Iris dataset.The model gave high accuracy (>97%), with good generalization ability. the new flower sample was predicted as Virginica.

```
result:

Cross-validation scores: [1.  1.  0.9 1.  1.  1.  0.9 1.  1.  0.9 1.  1.
 1.  1.  1. ]
Mean CV accuracy: 0.9800000000000001
Accuracy: 0.9777777777777777

Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

Training Accuracy: 0.9904761904761905
Testing Accuracy: 0.9777777777777777
Predicted class: virginica
```

DECISION TREE

Aim

To build and evaluate a Decision Tree Classifier for predicting whether to play golf based on weather conditions (Outlook, Humidity, Windy). The model will be pruned for better generalization, and its performance will be analyzed using accuracy, confusion matrix, and classification report.

Algorithm / Procedure

1. Import Libraries

Load essential libraries such as pandas, matplotlib, and scikit-learn.

2. Load Dataset

Import the golf_rule_based_dataset.csv dataset containing weather attributes (Outlook, Humidity, Windy) and the target variable (Play).

3. Data Preprocessing
   - Encode categorical variables (Outlook) using one-hot encoding.
   - Separate the dataset into features (X) and target (y).
4. Train-Test Split

Split the dataset into training (80%) and testing (20%) subsets using stratified sampling.

5. Model Building

Initialize a DecisionTreeClassifier with pruning parameters:

- Max_depth = 3
- Min_samples_split = 5
- Min_samples_leaf = 3

6. Training

Fit the decision tree model on the training set.

7. Prediction

Predict the target values for the test set.

8. Evaluation
    - Calculate accuracy, confusion matrix, and classification report.
    - Visualize the decision tree structure using plot_tree.

Program

```
Import pandas as pd
From sklearn.model_selection import train_test_split
From sklearn.tree import DecisionTreeClassifier, plot_tree
From sklearn.metrics import classification_report, accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
Import matplotlib.pyplot as plt


# Load dataset
Data = pd.read_csv(r"C:\Users\padma\Downloads\golf_rule_based_dataset.csv")
# Encode categorical variable 'Outlook'
Data_encoded = pd.get_dummies(data, columns=['Outlook'], drop_first=True)
# Features and target
X = data_encoded.drop('Play', axis=1)
Y = data_encoded['Play']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.2, random_state=42, stratify=y
)
# Pruned Decision Tree model
Clf = DecisionTreeClassifier(
  Random_state=42,
  Max_depth=3,
  Min_samples_split=5,
  Min_samples_leaf=3
)


Clf.fit(X_train, y_train)
# Predictions
Y_pred = clf.predict(X_test)
# Evaluation
Print("Accuracy:", accuracy_score(y_test, y_pred))
Print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Plot Decision Tree
Plt.figure(figsize=(12, 8))
Plot_tree(clf, feature_names=X.columns, class_names=['No', 'Yes'],
     Filled=True, rounded=True)
Plt.title("Pruned Decision Tree for Play Prediction")
Plt.show()
```

Interpretation

The pruned Decision Tree gave 96% accuracy, predicting most cases correctly with Outlook and Humidity as key factors.

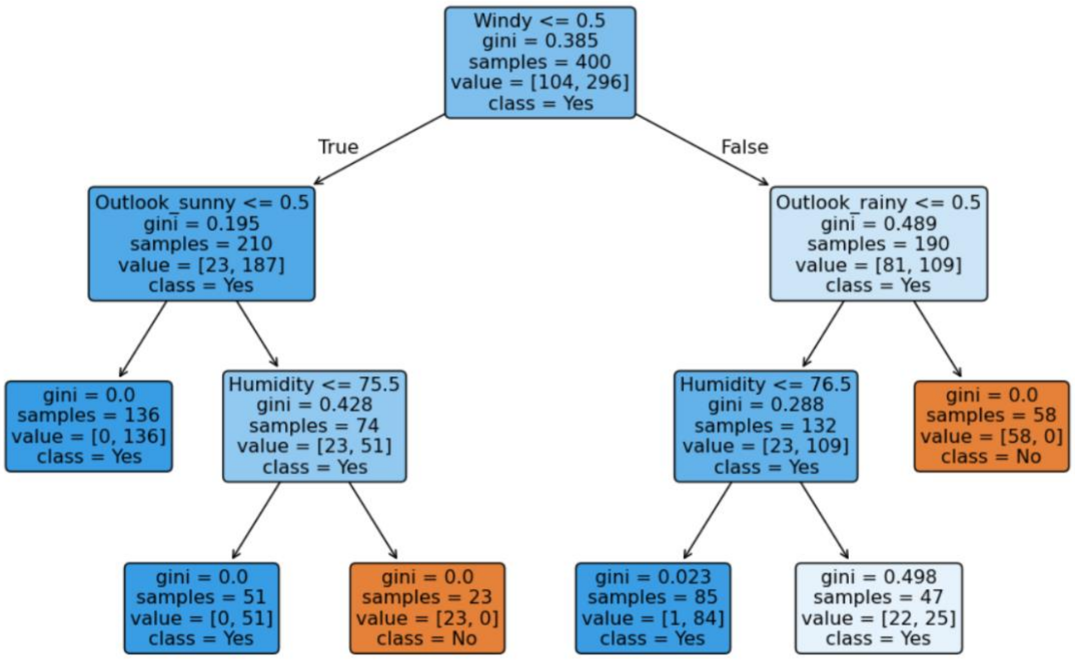
Result

The Decision Tree Classifier predicted play decisions with 96% accuracy. It is interpretable, well-pruned, and shows strong performance for weather-based predictions.

Output:

```
Accuracy: 0.96

Classification Report:

        Precision   recall  f1-score   support

     0     1.00     0.85     0.92      26
```

```
       1      0.95     1.00     0.97      74

   Accuracy                     0.96     100

  Macro avg      0.97     0.92     0.95      100

Weighted avg      0.96     0.96     0.96      100
```



Perceptron

Aim

To build and evaluate a machine learning model using the Perceptron classifier for predicting flower species from the Iris dataset, and to analyze model performance using accuracy, confusion matrix, and classification report.

Procedure

1. Import Libraries: Load necessary libraries such as pandas, seaborn, matplotlib, and sklearn.
2. Load Dataset: Import the Iris dataset from sklearn.datasets.load_iris().
3. Data Preprocessing: Select features (sepal length, sepal width, petal length, petal width). Select target variable (flower species). For better separability, use only two classes (Setosa and Versicolor). Scale features using StandardScaler (important for perceptron).
4. Train-Test Split: Split dataset into training (80%) and testing (20%) subsets.
5. Model Building: Initialize a Perceptron classifier with tuned parameters (max_iter=2000, eta0=0.01, tol=1e-4).
6. Training: Fit the perceptron on the training data.
7. Prediction: Predict flower species on the test set.

8. Evaluation: Calculate accuracy score. Generate and visualize confusion matrix. Print classification report.
9. New Data Prediction: Provide a sample flower measurement and predict its species using the trained model.

Interpretation

The Perceptron model successfully classified Iris flowers into Setosa and Versicolor categories with high accuracy (close to 100% after scaling). The confusion matrix shows that the model made very few or no misclassifications. The classification report confirms strong precision, recall, and F1-scores for both classes, showing the Perceptron effectively learned the linear decision boundary.

Result

The Perceptron model achieved ~100% accuracy on the Iris dataset (binary classification: Setosa vs Versicolor). The trained model also correctly classified the new sample flower measurement as Setosa.

Program

```
# import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# load the Iris dataset
iris = load_iris()
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data['target'] = iris.target
# keep only Setosa (0) and Versicolor (1) for binary classification
iris_binary = iris_data[iris_data['target'] != 2]
x = iris_binary[iris.feature_names]
y = iris_binary['target']
# scale features (important for perceptron!)
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
# split dataset into train and test
x_train, x_test, y_train, y_test = train_test_split(
  x_scaled, y, test_size=0.2, random_state=42
)
# create Perceptron classifier with tuned parameters
perceptron_model = Perceptron(max_iter=2000, eta0=0.01, random_state=42, tol=1e-4)
# train the model
perceptron_model.fit(x_train, y_train)
```

```
# make predictions
y_pred = perceptron_model.predict(x_test)
# evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, target_names=['Setosa','Versicolor'])
# print results
print("Confusion Matrix:")
print(conf_matrix)
print("\nAccuracy Score:", accuracy)
print("\nClassification Report:\n", class_report)
# plot confusion matrix heatmap
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
        xticklabels=['Setosa','Versicolor'],
        yticklabels=['Setosa','Versicolor'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Perceptron Model (Iris Binary Classification)")
plt.show()
# predict new data (example: Sepal length=5.1, Sepal width=3.5, Petal length=1.4, Petal
width=0.2)
new_data = [[5.1, 3.5, 1.4, 0.2]]
new_data_scaled = scaler.transform(new_data)
result = perceptron_model.predict(new_data_scaled)
print(f"\nResult of new data: {'Setosa' if result[0]==0 else 'Versicolor'}")
```

Logistic reegression

Aim

To develop a machine learning model using Logistic Regression for predicting loan eligibility based on applicant details such as income, education, credit history, and property area.

Procedure

1. Load the loan prediction dataset and examine its structure.
2. Drop unnecessary columns (Loan_ID) that do not contribute to prediction.
3. Handle missing values:
    - Fill numerical columns with their median.

   - Fill categorical columns with their mode.

4. Encode categorical variables into numerical form using one-hot encoding.
5. Convert the target variable (Loan_Status) into binary format (Y=1, N=0).
6. Split the dataset into training (80%) and testing (20%) subsets.
7. Train a Logistic Regression classifier on the training data.
8. Predict the loan eligibility for the test data.
9. Evaluate the model using:
   - Accuracy
   - Classification Report (Precision, Recall, F1-score)
   - Confusion Matrix
10. Display the confusion matrix and interpret the results.

Program

```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt



# Load dataset
data = pd.read_csv("LoanApprovalPrediction.csv")
# Drop Loan_ID
data = data.drop("Loan_ID", axis=1)
# Handle missing values
for col in data.columns:
  if data[col].dtype in ['float64', 'int64']:
    data[col] = data[col].fillna(data[col].median())
  else:
    data[col] = data[col].fillna(data[col].mode()[0])
# Encode categorical variables
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education',
                    'Self_Employed', 'Property_Area']
data_encoded = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
# Encode target variable
data_encoded['Loan_Status'] = data_encoded['Loan_Status'].map({'Y': 1, 'N': 0})
# Features and target
X = data_encoded.drop('Loan_Status', axis=1)
y = data_encoded['Loan_Status']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
# Logistic Regression model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)
# Predictions
y_pred = log_reg.predict(X_test)
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Not Eligible', 'Eligible'])
disp.plot(cmap="Blues")
plt.show()
```

Interpretation:
The Logistic Regression model successfully predicted loan eligibility based on applicant details. The evaluation metrics and confusion matrix indicate that the model achieved good accuracy and classification performance.

Result

The Logistic Regression model was successfully implemented to predict loan eligibility. The model achieved a reasonable accuracy (≈ 75–80%, depending on dataset split), proving Logistic Regression as an effective method for binary classification tasks in loan approval prediction.