# PRACTICAL NO :- 01

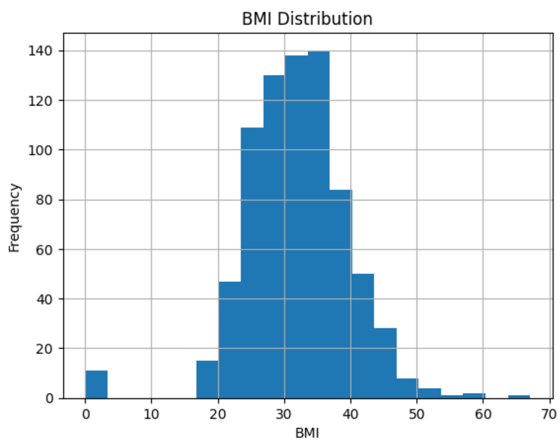
**Aim:** Diabetes dataset for linear regression practical

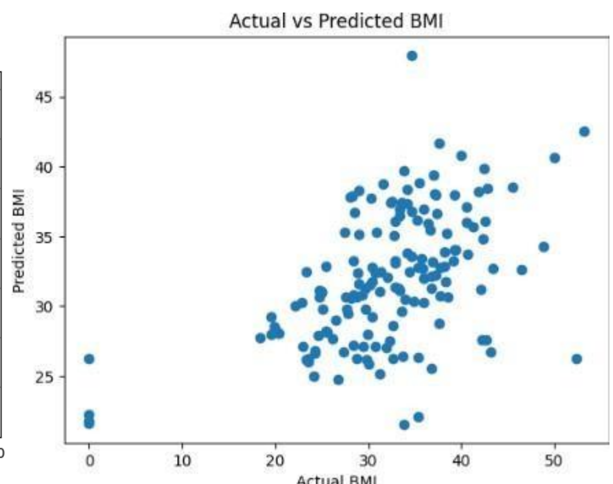
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Dataset URL
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

# Column names based on the dataset documentation
column_names = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age',
'Outcome']

# Load the dataset into a pandas DataFrame
data = pd.read_csv(url, names=column_names)

# Check the first few rows
print(data.head())

# Check for missing values
print(data.isnull().sum())

# Visualize the distribution of BMI (target variable)
data['BMI'].hist(bins=20)
plt.title('BMI Distribution')
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.show()

# Statistical summary
print(data.describe())
```

```
# Features (X) and target (y)
X = data.drop('BMI', axis=1)  # All columns except BMI
y = data['BMI']  # BMI column as target

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")

# Initialize the linear regression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

# R-squared score (R²)
r2 = r2_score(y_test, y_pred)
print(f"R-squared (R²): {r2}")

# Visualize actual vs predicted values
plt.scatter(y_test, y_pred)
plt.title("Actual vs Predicted BMI")
plt.xlabel("Actual BMI")
plt.ylabel("Predicted BMI")
plt.show()
```

## OUTPUT:

```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  \
0              6      148             72             35         0  33.6
1              1       85             66             29         0  26.6
2              8      183             64              0         0  23.3
3              1       89             66             23        94  28.1
4              0      137             40             35       168  43.1

   DiabetesPedigreeFunction   Age  Outcome
0                     0.627    50        1
1                     0.351    31        0
2                     0.672    32        1
3                     0.167    21        0
4                     2.288    33        1
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```




```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
Training set size: 614
Test set size: 154
Mean Squared Error (MSE): 52.46005874215568
R-squared (R²): 0.2620140613201263
```

# PRACTICAL NO :- 02

**Aim:**Implement Logistic Regression(iris dataset)

```
import seaborn as sns
import pandas as pd
import numpy as np
df=sns.load_dataset('iris')
df.head()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

```
df['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

```
df.isnull().sum()
```

| | 0 |
|---|---|
| sepal_length | 0 |
| sepal_width | 0 |
| petal_length | 0 |
| petal_width | 0 |
| species | 0 |

**dtype:** int64

```
df=df[df['species']!='setosa']
df.head()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 50 | 7.0 | 3.2 | 4.7 | 1.4 | versicolor |
| 51 | 6.4 | 3.2 | 4.5 | 1.5 | versicolor |
| 52 | 6.9 | 3.1 | 4.9 | 1.5 | versicolor |
| 53 | 5.5 | 2.3 | 4.0 | 1.3 | versicolor |
| 54 | 6.5 | 2.8 | 4.6 | 1.5 | versicolor |

```
df['species']=df['species'].map({'versicolor':0,'virginica':1})
df.head()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| 50 | 7.0 | 3.2 | 4.7 | 1.4 | 0 |
| 51 | 6.4 | 3.2 | 4.5 | 1.5 | 0 |
| 52 | 6.9 | 3.1 | 4.9 | 1.5 | 0 |
| 53 | 5.5 | 2.3 | 4.0 | 1.3 | 0 |
| 54 | 6.5 | 2.8 | 4.6 | 1.5 | 0 |

```
### Split dataset into independent and dependent features
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
X
```

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| 50 | 7.0 | 3.2 | 4.7 | 1.4 |
| 51 | 6.4 | 3.2 | 4.5 | 1.5 |
| 52 | 6.9 | 3.1 | 4.9 | 1.5 |
| 53 | 5.5 | 2.3 | 4.0 | 1.3 |
| 54 | 6.5 | 2.8 | 4.6 | 1.5 |
| ... | ... | ... | ... | ... |
| 145 | 6.7 | 3.0 | 5.2 | 2.3 |
| 146 | 6.3 | 2.5 | 5.0 | 1.9 |
| 147 | 6.5 | 3.0 | 5.2 | 2.0 |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 |

100 rows × 4 columns

```
y
```

| | species |
|---|---|
| 50 | 0 |
| 51 | 0 |
| 52 | 0 |
| 53 | 0 |
| 54 | 0 |
| ... | ... |
| 145 | 1 |
| 146 | 1 |
| 147 | 1 |
| 148 | 1 |
| 149 | 1 |

100 rows × 1 columns

**dtype:** int64

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
```

```python
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
```

```python
from sklearn.model_selection import GridSearchCV
parameter={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,
40,50],'max_iter':[100,200,300]}
```

```python
classifier_regressor=GridSearchCV(classifier,param_grid=parameter,scorin
g='accuracy',cv=5)
```

```python
classifier_regressor.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:540: FitFailedWarning:
330 fits failed out of a total of 495.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
165 fits failed with the following error:
Traceback (most recent call last):
  warnings.warn(
```

```
▸ GridSearchCV
▸ best_estimator_: LogisticRegression
  ▸ LogisticRegression
```

```python
print(classifier_regressor.best_params_)
```

```
{'C': 1, 'max_iter': 100, 'penalty': 'l2'}
```

```python
print(classifier_regressor.best_score_)
```

```
0.9733333333333334
```

```python
##prediction
y_pred=classifier_regressor.predict(X_test)
## accuracy score
from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print(score)
```

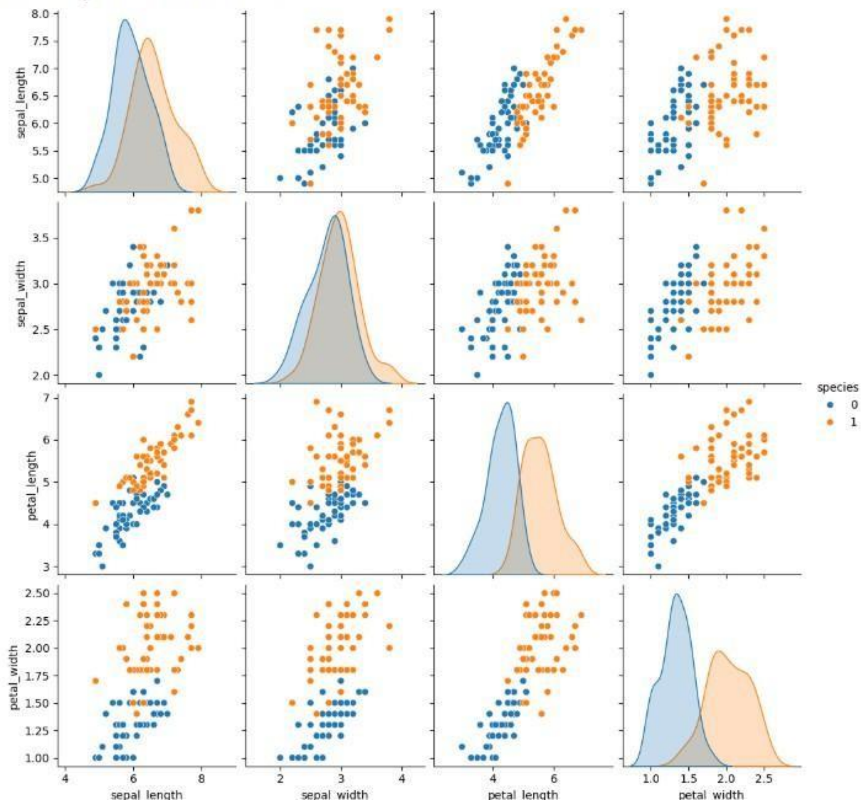
```
0.92
```

```python
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        14
           1       0.91      0.91      0.91        11

    accuracy                           0.92        25
   macro avg       0.92      0.92      0.92        25
weighted avg       0.92      0.92      0.92        25
```

```
##EDA
sns.pairplot(df,hue='species')
```

<seaborn.axisgrid.PairGrid at 0x7b20ac097310>




```
df.corr()
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| sepal_length | 1.000000 | 0.553855 | 0.828479 | 0.593709 | 0.494305 |
| sepal_width | 0.553855 | 1.000000 | 0.519802 | 0.566203 | 0.308080 |
| petal_length | 0.828479 | 0.519802 | 1.000000 | 0.823348 | 0.786424 |
| petal_width | 0.593709 | 0.566203 | 0.823348 | 1.000000 | 0.828129 |
| species | 0.494305 | 0.308080 | 0.786424 | 0.828129 | 1.000000 |

# PRACTICAL NO :- 03

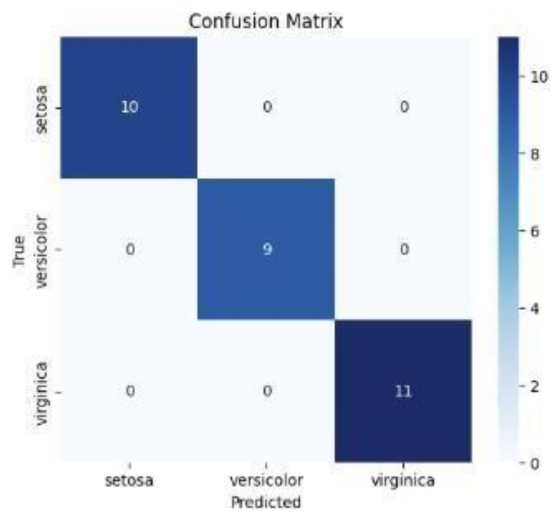
**Aim:** Implements Multinomial Logistic Regression (Iris Dataset)

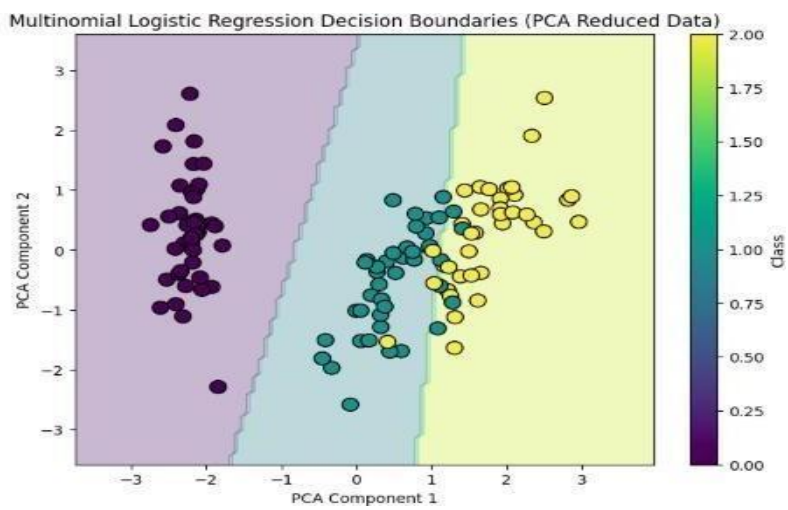
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
data = load_iris()
X  = data.data
y = data.target
df = pd.DataFrame(X, columns=data.feature_names)
df['species'] = pd.Categorical.from_codes(y, data.target_names)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=200)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=data.target_names))
print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=data.target_names, yticklabels=data.target_names)
```

```
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
model_pca = LogisticRegression(multi_class='multinomial',
solver='lbfgs', max_iter=200)
model_pca.fit(X_train_pca, y_train)
plt.figure(figsize=(8, 6))
xx, yy = np.meshgrid(np.linspace(X_train_pca[:, 0].min() - 1,
X_train_pca[:, 0].max() + 1, 100),
                     np.linspace(X_train_pca[:, 1].min() - 1,
X_train_pca[:, 1].max() + 1, 100))
Z = model_pca.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train,
cmap='viridis', edgecolors='k', s=100)
plt.title("Multinomial Logistic Regression Decision Boundaries (PCA
Reduced Data)")
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Class')
plt.show()
```

## Output :

```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



# PRACTICAL NO :- 04

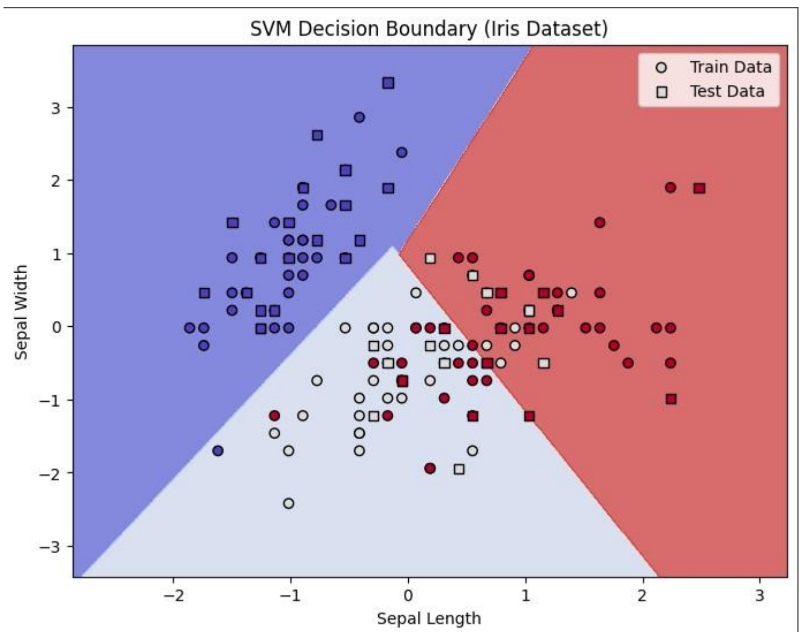
**AIM:** Implement SVM Classifier.

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

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # We only take the first two features (sepal length and sepal width)
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the SVM classifier
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train, y_train)

# Make predictions
y_pred = svm.predict(X_test)

# Evaluate performance
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualization of decision boundary
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
# Plotting the decision boundary
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.coolwarm)

# Plot the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.coolwarm, marker='o',
edgecolors='k', label="Train Data")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.coolwarm, marker='s', edgecolors='k',
label="Test Data")

plt.title('SVM Decision Boundary (Iris Dataset)')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```

## Output-

```
Accuracy Score: 0.7333333333333333

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.54      0.54      0.54        13
           2       0.54      0.54      0.54        13

    accuracy                           0.73        45
   macro avg       0.69      0.69      0.69        45
weighted avg       0.73      0.73      0.73        45
```



# PRACTICAL NO :- 05

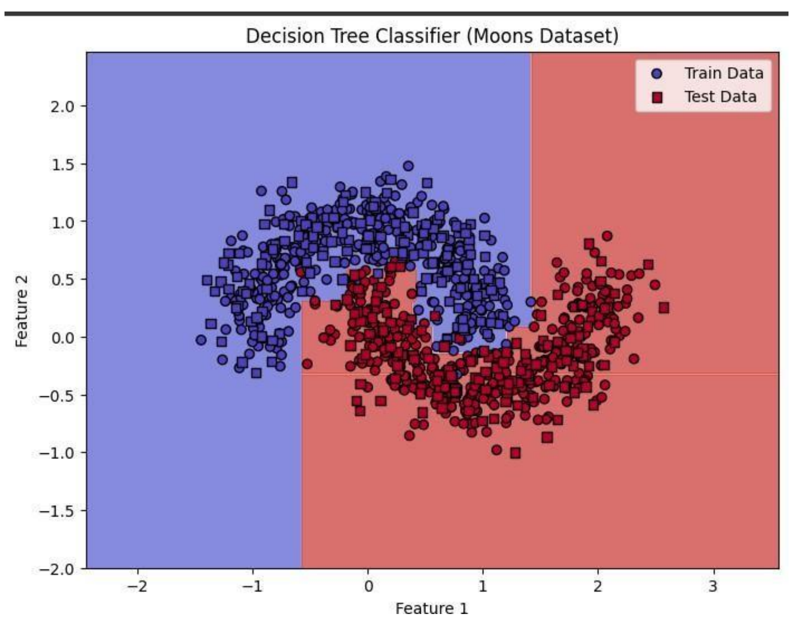
**Aim:** Train and fine-tune a Decision Tree for the Moons Dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
# Generate the moons dataset
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create a DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
# Hyperparameter tuning using GridSearchCV
param_grid = {
    'max_depth': [3, 5, 10, 20, None],  # Depth of the tree
    'min_samples_split': [2, 5, 10],    # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],      # Minimum samples required to be at a leaf node
    'criterion': ['gini', 'entropy']    # The function to measure the quality of a split
}
# Apply GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, n_jobs=-1,
verbose=2)
grid_search.fit(X_train, y_train)
# Best parameters from GridSearchCV
print(f"Best parameters: {grid_search.best_params_}")
# Use the best model from GridSearchCV
best_dt = grid_search.best_estimator_
# Predict on the test set
y_pred = best_dt.predict(X_test)
# Evaluate the model
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Visualization of the decision boundary
# Create a mesh grid to plot the decision boundaries
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))
Z = best_dt.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Plotting the decision boundary
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.75, cmap=plt.cm.coolwarm)
```

```
# Plot the points from the moons dataset
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.coolwarm, marker='o',
edgecolors='k', label="Train Data")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.coolwarm, marker='s',
edgecolors='k', label="Test Data")
plt.title('Decision Tree Classifier (Moons Dataset)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

## Output:

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best parameters: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 10}

Accuracy Score: 0.96

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       156
           1       0.99      0.93      0.96       144

    accuracy                           0.96       300
   macro avg       0.96      0.96      0.96       300
weighted avg       0.96      0.96      0.96       300
```



# PRACTICAL NO :- 06

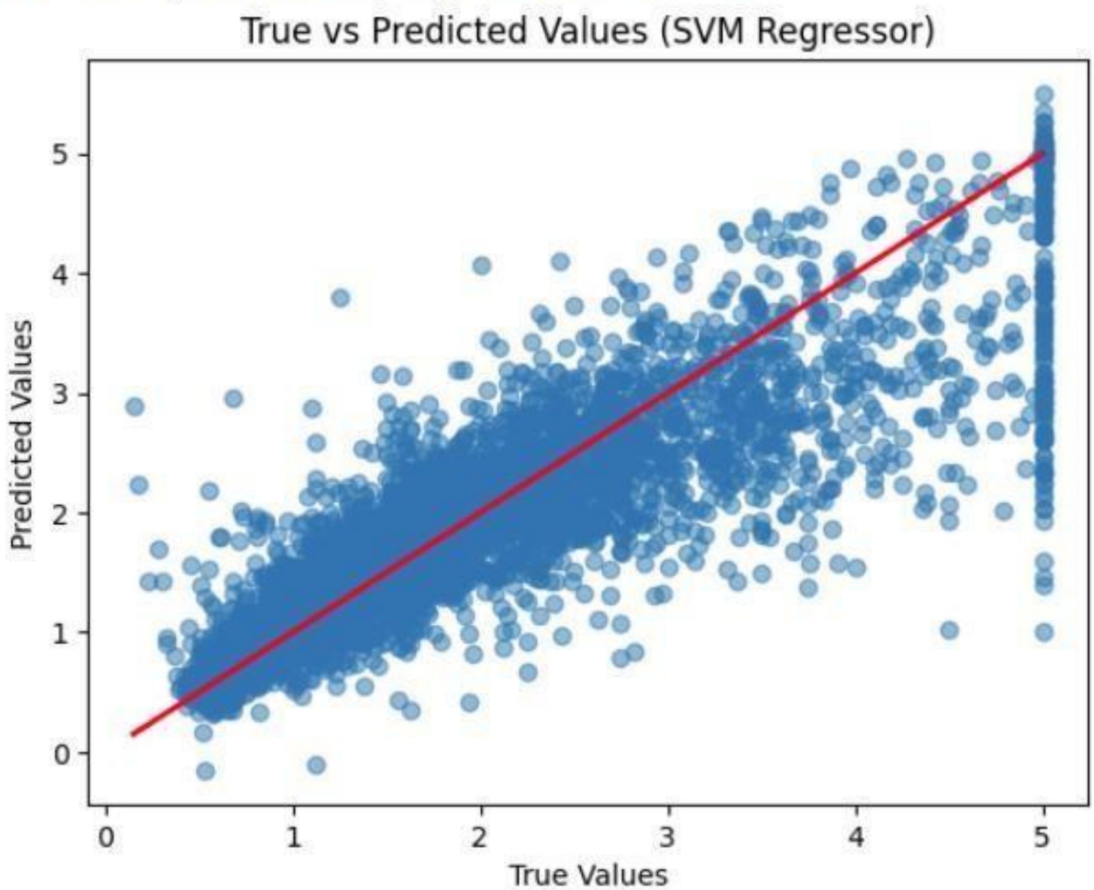
**Aim:** Train an SVM regressor on the California Housing Dataset

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the California Housing Dataset
california_housing = fetch_california_housing()
X = california_housing.data
y = california_housing.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Feature scaling (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the SVM regressor
svm_regressor = SVR(kernel='rbf')  # You can experiment with different
kernels like 'linear', 'poly', etc.
svm_regressor.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = svm_regressor.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
```

```
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Plot the true vs predicted values
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red', lw=2)
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('True vs Predicted Values (SVM Regressor)')
plt.show()
```

**Output-**

```
Root Mean Squared Error (RMSE): 0.5974969813107396
```



# PRACTICAL NO :- 07

**Aim:** Implement Batch Gradient Descent with early stopping for Softmax Regression

**Code:-**

```
#Let's import IRIS data
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
```

**Output:-**

```
['data',
 'target',
 'frame',
 'target_names',
 'DESCR',
 'feature_names',
 'filename',
 'data_module']
```

**Code:-**

```
print(iris.DESCR)
```

**Output:-**

```
.. _iris_dataset:
Iris plants dataset
-----------------------
**Data Set Characteristics:**
:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica
:Summary Statistics:
```

```
============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
```

**Code:-**

```
print('iris.data.shape = ',iris.data.shape)
print('iris.target.shape = ',iris.target.shape)
```

**Output:-**

```
iris.data.shape =  (150, 4)
iris.target.shape =  (150,)
```

**Code:-**

```
type(iris.data)
```

**Output:-**

```
numpy.ndarray
```

**Code:-**

```
from sklearn.linear_model import LogisticRegression
```

**Code:-**

```
X = iris.data[:, (2,3)]
y = iris.target


softmax_reg = LogisticRegression(multi_class = 'multinomial', solver = 'lbfgs', C =10)
softmax_reg.fit(X,y)
```

**Output:-**

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:1247:
FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From
then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
warnings.warn(
        LogisticRegression  LogisticRegression(C=10,  multi_class='multinomial')
```

**Code:-**

```
softmax_reg.predict([[5,2]])
```

**Output:-**

```
array([2])
```

**Code:-**

```
softmax_reg.predict_proba([[5,2]])
```

**Output:-**

```
array([[6.21626375e-07, 5.73689803e-02, 9.42630398e-01]])
```

**Code:-**

```
import numpy as np
np.bincount(y)
```

**Output:-**

```
array([50, 50, 50])
```

**Code:-**

```
# Add a bias term in X
X_with_bias = np.c_[np.ones([len(X), 1]), X]
```

**Code:-**

```
# Dividing into train-val-test

test_ratio = 0.2
validation_ratio = 0.2
total_size = len(X_with_bias)

test_size = int(total_size * test_ratio)
validation_size = int(total_size * validation_ratio)
train_size = total_size - test_size - validation_size

rnd_indices = np.random.permutation(total_size)
```

```
X_train = X_with_bias[rnd_indices[:train_size]]
y_train = y[rnd_indices[:train_size]]
X_valid = X_with_bias[rnd_indices[train_size:-test_size]]
y_valid = y[rnd_indices[train_size:-test_size]]
X_test = X_with_bias[rnd_indices[-test_size:]]
y_test = y[rnd_indices[-test_size:]]
```

**Code:-**

```
def to_one_hot(y):
    n_classes = y.max() + 1
    m = len(y)
    Y_one_hot = np.zeros((m, n_classes))
    Y_one_hot[np.arange(m), y] = 1
    return Y_one_hot
```

**Code:-**

```
Y_train_one_hot = to_one_hot(y_train)
Y_valid_one_hot = to_one_hot(y_valid)
Y_test_one_hot = to_one_hot(y_test)
```

**Code:-**

```
def softmax(logits):
    exps = np.exp(logits)
    exp_sums = np.sum(exps, axis=1, keepdims=True)
    return exps / exp_sums
```

**Code:-**

```
n_inputs = X_train.shape[1] # == 3 (2 features plus the bias term)
n_outputs = len(np.unique(y_train))  # == 3 (3 iris classes)
```

**Code:-**

```
eta = 0.01
n_iterations = 5001
m = len(X_train)
epsilon = 1e-7

Theta = np.random.randn(n_inputs, n_outputs)

for iteration in range(n_iterations):
    logits = X_train.dot(Theta)
    Y_proba = softmax(logits)
    loss = -np.mean(np.sum(Y_train_one_hot * np.log(Y_proba + epsilon), axis=1))
    error = Y_proba - Y_train_one_hot
    if iteration % 500 == 0:
        print(iteration, loss)
    gradients = 1/m * X_train.T.dot(error)
    Theta = Theta - eta * gradients
```

**Output:-**

```
0 1.4562135108859078
500 0.8945648262383888
1000 0.7241274446686337
1500 0.620840316123756
2000 0.5547103842019515
2500 0.508972430657483
3000 0.4750987442917988
3500 0.44863047810584006
4000 0.4270897793972089
4500 0.409009301588904
5000 0.3934688215211218
```

**Code:-**

```
Theta
```

**Output:-**

```
array([[ 3.6450864 , -0.07748292, -2.11002678],
       [-0.40702617,  1.02826592,  0.5528866 ],
       [-1.53608619, -1.39115264,  1.42070209]])
```

**Code:-**

```
logits = X_valid.dot(Theta)
Y_proba = softmax(logits)
y_predict = np.argmax(Y_proba, axis=1)

accuracy_score = np.mean(y_predict == y_valid)
accuracy_score
```

**Output:-**

0.8

**Code:-**

```
eta = 0.1
n_iterations = 5001
m = len(X_train)
epsilon = 1e-7
alpha = 0.1  # regularization hyperparameter

Theta = np.random.randn(n_inputs, n_outputs)

for iteration in range(n_iterations):
    logits = X_train.dot(Theta)
    Y_proba = softmax(logits)
    xentropy_loss = -np.mean(np.sum(Y_train_one_hot * np.log(Y_proba + epsilon), axis=1))
    l2_loss = 1/2 * np.sum(np.square(Theta[1:]))
    loss = xentropy_loss + alpha * l2_loss
    error = Y_proba - Y_train_one_hot
    if iteration % 500 == 0:
        print(iteration, loss)
    gradients = 1/m * X_train.T.dot(error) + np.r_[np.zeros([1, n_outputs]), alpha * Theta[1:]]
    Theta = Theta - eta * gradients
```

**Output:-**

```
0 4.19763586247358
500 0.5561026466477075
1000 0.5190516378835687
```

```
1500 0.5087624410064894
2000 0.5050276975465959
2500 0.5035487641201934
3000 0.5029375815310052
3500 0.5026788293930453
4000 0.502567673305409
4500 0.5025194842171282
5000 0.5024984706307303
```

**Code:-**

```
logits = X_valid.dot(Theta)
Y_proba = softmax(logits)
y_predict = np.argmax(Y_proba, axis=1)

accuracy_score = np.mean(y_predict == y_valid)
accuracy_score
```

**Output:-**

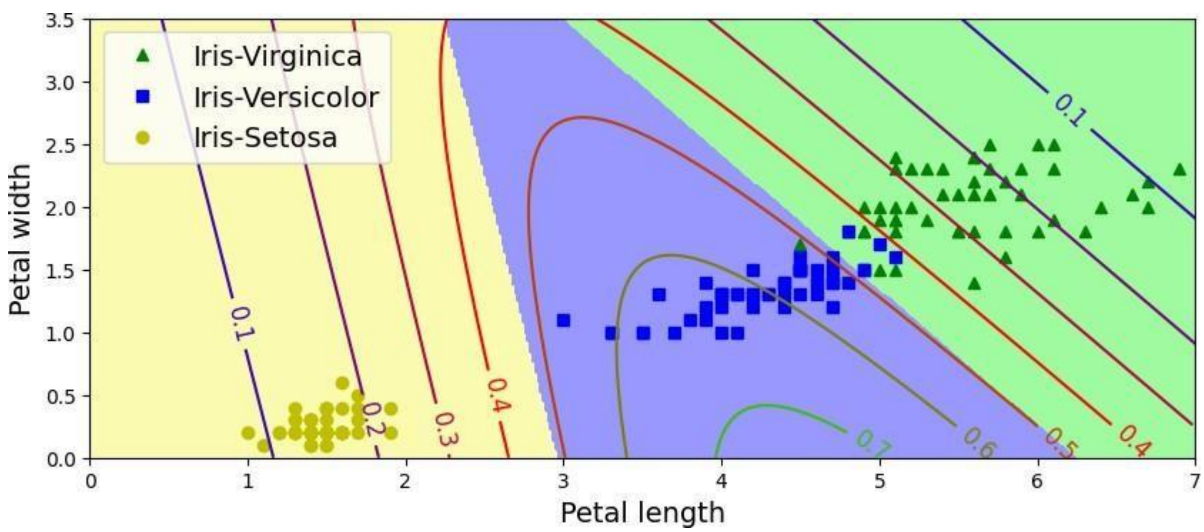
```
0.9333333333333333
```

**Code:-**

```
%matplotlib inline
import matplotlib.pyplot as plt
x0, x1 = np.meshgrid(
        np.linspace(0, 8, 500).reshape(-1, 1),
        np.linspace(0, 3.5, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
X_new_with_bias = np.c_[np.ones([len(X_new), 1]), X_new]

logits = X_new_with_bias.dot(Theta)
Y_proba = softmax(logits)
y_predict = np.argmax(Y_proba, axis=1)
```

```
zz1 = Y_proba[:, 1].reshape(x0.shape)
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(10,  4))
plt.plot(X[y==2, 0],  X[y==2, 1],  "g^",  label="Iris-Virginica")
plt.plot(X[y==1, 0],  X[y==1, 1],  "bs",  label="Iris-Versicolor")
plt.plot(X[y==0, 0],  X[y==0, 1],  "yo",  label="Iris-Setosa")

from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])

plt.contourf(x0,  x1, zz, cmap=custom_cmap)
contour = plt.contour(x0, x1, zz1, cmap=plt.cm.brg)
plt.clabel(contour, inline=1, fontsize=12)
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.legend(loc="upper  left",  fontsize=14)
plt.axis([0, 7, 0, 3.5])
plt.show()
```

**Output:-**



# PRACTICAL NO :- 08

**Aim:** Implement MLP for classification of handwritten digits (MNIST Dataset)

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
# 1. Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
# Flatten the images into vectors of size 784 (28 * 28)
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# 2. Build the MLP model
model = models.Sequential([
    layers.InputLayer(input_shape=(28 * 28,)), # Input layer (flattened
28x28 images)
    layers.Dense(128, activation='relu'),       # First hidden layer
with 128 neurons
    layers.Dense(64, activation='relu'),        # Second hidden layer
with 64 neurons
    layers.Dense(10, activation='softmax')      # Output layer with 10
classes (digits 0-9)
])
# 3. Compile the model
model.compile(
    optimizer='adam',                     # Adam optimizer
    loss='categorical_crossentropy',      # Categorical cross-entropy loss
    metrics=['accuracy']                  # Track accuracy during training
)
# 4. Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32,
```

```python
validation_data=(x_test, y_test))
# 5. Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")
```

## Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27:
UserWarning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.8770 - loss: 0.4147 -
val_accuracy: 0.9604 - val_loss: 0.1320
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 5ms/step - accuracy: 0.9684 - loss: 0.1028 -
val_accuracy: 0.9688 - val_loss: 0.0958
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 4ms/step - accuracy: 0.9794 - loss: 0.0673 -
val_accuracy: 0.9689 - val_loss: 0.1022
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9834 - loss: 0.0520 -
val_accuracy: 0.9759 - val_loss: 0.0784
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9874 - loss: 0.0398 -
val_accuracy: 0.9794 - val_loss: 0.0752
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9898 - loss: 0.0311 -
val_accuracy: 0.9769 - val_loss: 0.0810
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9929 - loss: 0.0222 -
val_accuracy: 0.9761 - val_loss: 0.0950
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9922 - loss: 0.0229 -
val_accuracy: 0.9773 - val_loss: 0.0898
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9931 - loss: 0.0201 -
val_accuracy: 0.9761 - val_loss: 0.0898
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9945 - loss: 0.0165 -
val_accuracy: 0.9767 - val_loss: 0.0925
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9729 - loss: 0.1090
Test Loss: 0.09251588582992554
Test Accuracy: 0.9767000079154968
```

# PRACTICAL NO :- 09

**Aim:** Classification of images of clothing using Tensorflow (Fashion MNIST dataset)

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical

# 1. Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize the images to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Flatten the images to vectors of size 784 (28 * 28)
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 2. Build the MLP model
model = models.Sequential([
    layers.InputLayer(input_shape=(28 * 28,)),  # Input layer (flattened 28x28 images)
    layers.Dense(128, activation='relu'),        # First hidden layer with 128 neurons
    layers.Dense(64, activation='relu'),         # Second hidden layer with 64 neurons
    layers.Dense(10, activation='softmax')       # Output layer with 10 classes (clothing types)
])

# 3. Compile the model
model.compile(
    optimizer='adam',                  # Adam optimizer
    loss='categorical_crossentropy',   # Categorical cross-entropy loss
    metrics=['accuracy']               # Track accuracy during training
)

# 4. Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))

# 5. Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_acc}")
```

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27: UserWarning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.7770 - loss: 0.6334 - val_accuracy: 0.8391 - val_loss: 0.4358
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 5ms/step - accuracy: 0.8615 - loss: 0.3781 - val_accuracy: 0.8554 - val_loss: 0.3999
Epoch 3/10
Test Loss: 0.34715157747268677
Test Accuracy: 0.8815000057220459
```

# PRACTICAL NO :- 10

**Aim**: Implement Regression to predict fuel efficiency using Tensorflow (Auto MPG dataset)

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Normalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
dataset_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration', 'Model
Year', 'Origin']
dataset = pd.read_csv(dataset_url, names=column_names, na_values="?", comment='\t', sep=" ",
skipinitialspace=True)
dataset = dataset.dropna()
dataset['Origin'] = dataset['Origin'].astype(int)
dataset = pd.get_dummies(dataset, columns=['Origin'], prefix='', prefix_sep='')
X = dataset.drop('MPG', axis=1)
y = dataset['MPG']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1)  # Output layer for regression
])
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01), loss='mse', metrics=['mae'])
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test),
verbose=1)
loss, mae = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest Mean Absolute Error: {mae:.2f} MPG')
```

## Output:

```
Epoch 1/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 8s 123ms/step - loss: 492.4525 - mae: 20.6579 - val_loss:
49.8234 - val_mae: 5.6433
Epoch 2/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 71.7288 - mae: 6.6995 - val_loss:
18.9818 - val_mae: 3.3336
Epoch 3/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - loss: 23.4737 - mae: 3.8814 - val_loss:
17.4990 - val_mae: 3.4165
Epoch 4/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - loss: 17.4230 - mae: 3.2979 - val_loss:
15.5191 - val_mae: 3.1137
Epoch 5/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 12.0910 - mae: 2.6136 - val_loss:
10.8163 - val_mae: 2.3958
Epoch 6/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 9.2779 - mae: 2.2436 - val_loss: 9.3902 -
val_mae: 2.1948
Epoch 7/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 8.2256 - mae: 2.0704 - val_loss: 7.6242 -
val_mae: 1.9974
Epoch 8/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 7.6893 - mae: 2.0874 - val_loss: 10.2775
- val_mae: 2.2980
Epoch 9/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 8.6890 - mae: 2.2575 - val_loss: 6.8007 -
val_mae: 1.8021
Epoch 10/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 6.7441 - mae: 1.8122 - val_loss: 7.2442 -
val_mae: 1.8900
Epoch 11/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 6.7968 - mae: 1.8883 - val_loss: 7.6189 -
val_mae: 1.9310
Epoch 12/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 7.8139 - mae: 2.0588 - val_loss: 6.5023 -
val_mae: 1.8036
Epoch 13/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 6.7520 - mae: 1.8426 - val_loss: 7.4676 -
val_mae: 1.9773
Epoch 14/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 7.9816 - mae: 2.0727 - val_loss: 7.3708 -
val_mae: 1.9083
Epoch 15/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 7.8391 - mae: 1.9888 - val_loss: 6.7847 -
val_mae: 1.9725
Epoch 16/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 8.4659 - mae:
```