FACULTY OF ENGINEERING AND TECHNOLOGY

JAIN DEEMED-TO-BE UNIVERSITY

Department of Computer Science & Engineering
(Artificial Intelligence and Machine Learning)

LAB RECORD

21CS5AM09L – Advanced Machine Learning LAB

(Batch 2021-2025)

July - December-2023
Global Campus

Jakkasandra Post, Kanakapura Taluk,
Ramanagara District - Pin Code: 562 112

SCHOOL OF ENGINEERING AND TECHNOLOGY

Department of Computer Science & Engineering (Artificial Intelligence and Machine Learning)

Global Campus

Jakkasandra Post, Kanakapura Taluk, Ramanagara District - Pin Code: 562 112

21CS5AM08L – DATA WAREHOUSING AND MINING LAB

2023-2024

Register No: 21BTRCL059

It is hereby certified that this is the bona fide record of work done by Mr./Ms. M.SHRAVANI PRIYA during the academic year 2023-2024 and submitted for the University Practical Examination held on 27-11-20203.

Over All Marks

Staff-in-charge **Head of the Department**

Internal Examiner **External Examiner**

TABLE OF CONTENTS					
#	**Date**	**Title of the Experiment**	**Page**	**Marks**	**Signatue of Faculty**
1.		Write a program to implement a logistic regression with 2 different datasets and evaluate the classification accuracy.			
2.		Write a program to implement a support vector machine with various datasets and evaluate the classification accuracy			
3.		Write a program to implement a K- Nearest Neigbour algorithm with 2 different datasets and evaluate the classification accuracy.			
4.		Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.			
5.		Write a program to implement the Naïve Bayes classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.			
6.		Write a program to implement hierarchical clustering with two different datasets			
7.		Write a program to implement K- Mean clustering algorithm with two different datasets			
8.		Using feature forward selection approaches, reduce the dimensionality of the KDD cup99 dataset.			
9.		Using feature backward elimination approaches, reduce the dimensionality of the KDD cup99 dataset.			
10.		Write a program to implement DBSACN algorithm with two different datasets			

Experiment: 1 **Date:**

Aim: To implement logistic regression with 2 different datasets and evaluate the classification accuracy.

Procedure:

Data Loading and Inspection:

- Import the dataset using Pandas and inspect the first few rows, data types, and missing values.

Data Preprocessing:

- Extract features (X) and labels (Y) from the dataset. Handle missing values if any.
- Encode categorical labels using Label Encoder.

Data Splitting:

- Split the dataset into training and testing sets for model evaluation.

Logistic Regression Model Training and Evaluation:

- Create a logistic regression model and train it on the training set.
- Evaluate the model on the test set and assess classification accuracy.

Source Code:

```
import pandas as pd
import numpy as np
from pandas_profiling import ProfileReport
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import mode
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
data3=pd.read_csv(“iris.csv")
data3.head()
data3.info()
```

```
from sklearn.preprocessing import LabelEncoder
X= data3.iloc[:,1:5]
Y = data3.iloc[:,-1]

label_encoder = LabelEncoder()
label = label_encoder.fit_transform(Y)
Y= label_encoder.transform(Y)
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=2)
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(X_train,Y_train)
y_pred=reg.predict(X_test)
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
accuracy = accuracy_score(Y_test,y_pred)
accuracy
```

Output:

```
In [18]:   1 y_pred
Out[18]: array([0, 0, 2, 0, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 1, 1, 0, 1, 2, 1, 2, 1,
                2, 1, 1, 0, 0, 2, 0, 2])

In [16]:   1 from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score,f1_score
           2 accuracy = accuracy_score(Y_test,y_pred)

In [17]:   1 accuracy
Out[17]: 0.9666666666666667
```

Result:

Logistic Regression was implemented and the results were noted.

Experiment: 2 **Date:**

Aim: To implement support vector machine with various datasets and evaluate the classification accuracy.

Procedure:

Data Loading and Inspection:

- Import required libraries and read the Iris dataset.
- Display the first few rows of the dataset.

Data Preprocessing:

- Separate features (X) and labels (Y) from the dataset.
- Split the data into training and testing sets.

Support Vector Classification:

- Create a linear SVM classifier, fit it on the training set, and make predictions on the test set.
- Evaluate the classification performance using metrics such as accuracy, confusion matrix, and classification report.

Support Vector Regression:

- Load the Real Estate dataset and prepare features (x) and target variable (y).
- Split the data into training and testing sets.

Support Vector Regression Evaluation:

- Create an SVR model, fit it on the training set, and predict on the test set.
- Evaluate the regression performance using metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
df = pd.read_csv(“iris.csv")
df.head()
Y = df['Species']
X = df.drop(['Id','Species'], axis=1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)

from sklearn.svm import SVC
svc = SVC(kernel='linear',C=1E10)
svc.fit(X_train, Y_train)
y_pred = svc.predict(X_test)
svc.score(X_test, Y_test)
svc.support_vectors_
from sklearn.metrics import classification_report
print(classification_report(Y_test, y_pred))
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(Y_test,y_pred)
ac=accuracy_score(Y_test, y_pred)
print(cm)
print(ac)

# Support Vector Regression
# svm for regression.
import pandas as pd
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
df = pd.read_csv("Real estate.csv")
y = df.pop("Y house price of unit area")
df.head()
x = df.drop(['No'], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
svr = SVR()
svr.fit(x_train, y_train)
y_pred = svr.predict(x_test)
```

```
from sklearn.metrics import r2_score
result = r2_score(y_test, y_pred)
print(result)

#If True returns MSE value, if False returns RMSE value.
from sklearn.metrics import mean_squared_error
result=  mean_squared_error(y_test,y_pred,squared=False)
result1=   mean_squared_error(y_test,y_pred,squared=True)
print(result)
print(result1)
```

Output:

```
In [7]:
from sklearn.metrics import classification_report
print(classification_report(Y_test, y_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       0.79      0.94      0.86        16
 Iris-virginica       0.88      0.64      0.74        11

       accuracy                           0.89        45
      macro avg       0.89      0.86      0.86        45
   weighted avg       0.89      0.89      0.88        45
```

```
In [8]:
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(Y_test,y_pred)
ac=accuracy_score(Y_test, y_pred)
print(cm)
print(ac)
```

```
[[18  0  0]
 [ 0 15  1]
 [ 0  4  7]]
0.8888888888888888
```

Result:

Support Vector Machine and Support Vector Regression was implemented and the results were noted.

Experiment: 3 **Date:**

Aim: To implement K- Nearest Neighbor algorithm with 2 different datasets and evaluate the classification accuracy.

Procedure:

Data Loading and Inspection:

- Import required libraries and read the first dataset (tested.csv).[Titanic Dataset]
- Display the first few rows of the dataset

Data Preprocessing:

- Separate features (x) and labels (y) from the dataset.
- Drop irrelevant columns and encode categorical variables.

Data Splitting:

- Split the dataset into training and testing sets.
- K-Nearest Neighbours (KNN) Model Training and Evaluation:

Create a KNN classifier, fit it on the training set, and predict on the test set.

- Evaluate the classification performance using metrics such as accuracy and confusion matrix.

Prediction on New Data:

- Make predictions on new data points using the trained KNN model.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
df=pd.read_csv(r"tested.csv")
df.head()
from sklearn.preprocessing import LabelEncoder
y = df['Survived']
x = df.drop(['Survived'],axis=1)data3=pd.read_csv("iris.csv")
x.head()
```

```
x=x.drop(['Name'],axis=1)
x=x.drop(['Ticket'],axis=1)
x=x.drop(['SibSp'],axis=1)
x=x.drop(['Parch'],axis=1)
x=x.drop(['Cabin'],axis=1)
x=x.drop([‘Embarked'],axis=1)

from sklearn.preprocessing import LabelEncoder
# Create a LabelEncoder instance
label_encoder = LabelEncoder()
# Encode the 'Sex' column
x['Sex'] = label_encoder.fit_transform(x['Sex'])
x.head()
x.fillna("0")
x.isna().sum()
mean_age = x['Age'].mean()
x['Age'].fillna(mean_age, inplace=True)
mean_age = x['Fare'].mean()
x['Fare'].fillna(mean_age, inplace=True)
import numpy as np
x.replace([np.inf, -np.inf], 1e10, inplace=True)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size= 0.2 )
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=4)
knn1.fit(x_train,y_train)
y_pred = knn1.predict(x_test)
y_pred
from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_pred,y_test)
print(cm)
result=knn1.predict([[3,1,34.5,7.8292]])
result
knn1.predict_proba([[3,1,34,7.05]])
```

Output:

```
In [46]:  y_pred = knn1.predict(x_test)
          y_pred

Out[46]: array([0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int64)

In [47]:  from sklearn.metrics import accuracy_score
          accuracy_score(y_pred,y_test)

Out[47]: 0.7142857142857143

In [48]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_pred,y_test)
          print(cm)

         [[49 21]
          [ 3 11]]

In [49]:  result=knn1.predict([[3,1,34.5,7.8292]])
          result

Out[49]: array([0], dtype=int64)

In [51]:  knn1.predict_proba([[3,1,34,7.05]])

Out[51]: array([[1., 0.]])

In [ ]:
```

Result:

K Nearest Neighbor was implemented and the results were noted.

Experiment: 4 **Date:**

Aim: To implement and demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Procedure:

Data Loading and Inspection:

- Import required libraries and read the Iris dataset.
- Display the first few rows of the dataset and check for information.

Data Preprocessing:

- Separate features (x) and labels (y) from the dataset.
- Drop irrelevant columns ('Id').

Decision Tree Model Training:

- Create a Decision Tree classifier with entropy as the criterion.
- Fit the model on the features and labels.

Decision Tree Visualization:

- Visualize the trained Decision Tree.

Prediction with Decision Tree:

- Make a prediction using the trained Decision Tree model for a new data point.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
data1 = pd.read_csv("iris.csv")
data1.head()
y = data1['Species']
x = data1.drop('Species', axis=1)
x=x.drop('Id',axis=1)
```

data1.info()

x.head()

x.shape

x.head()

x.shape

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion='entropy')

dt.fit(x,y)

from sklearn import tree

dtree = tree.plot_tree(dt, filled=True)

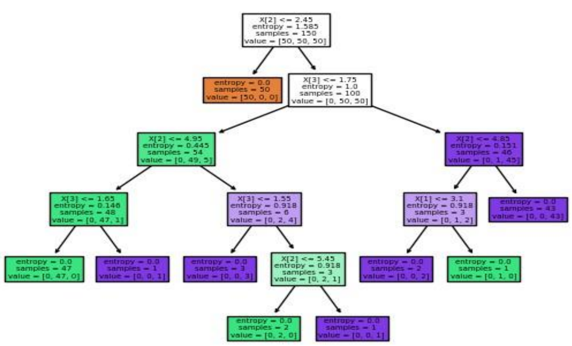
#outlook,temperature,humidity,windy

dt.predict([[1,2,1,0]])

Output:

```
In [16]:
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='entropy')

dt.fit(x,y)

Out[16]: DecisionTreeClassifier(criterion='entropy')
```

```
In [17]:
from sklearn import tree
dtree = tree.plot_tree(dt, filled=True)
```




```
In [18]:
#outlook,temperature,humidity,windy
dt.predict([[1,2,1,0]])

Out[18]: array(['Iris-setosa'], dtype=object)
```

Result:

Decision Tree based ID3 algorithm was implemented and the results were noted.

Experiment: 5 **Date:**

Aim: To implement the Naïve Bayes classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Procedure:

Data Loading and Inspection:

- Read the zoo dataset.
- Encode categorical variable ('animal_name') using LabelEncoder.

Data Preprocessing:

- Split the dataset into inputs and target.
- Split the data into training and testing sets.

Naive Bayes Model Training:

- Create and train a Gaussian Naive Bayes classifier using the training set.

Prediction and Confusion Matrix:

- Make predictions on the test set.
- Calculate and print the confusion matrix.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
# Load the dataset
df = pd.read_csv(r" zoo.csv”)
```

```
# Encode the 'animal_name' column
le = LabelEncoder()
df['animal_name'] = le.fit_transform(df['animal_name'])
# Split the data into inputs and target
inputs = df.drop('class_type', axis='columns')
target = df[‘class_type']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.2, random_state=42)
# Create and train the Gaussian Naive Bayes classifier on the training set
classifier = GaussianNB()
classifier.fit(X_train, y_train)
# Make predictions on the test set
y_pred = classifier.predict(X_test)
# Calculate and print the confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

Output:

```
# Calculate and print the confusion matrix
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
print(confusion_matrix)

[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  1]]

In [20]:  y_pred = classifier.predict(X_test)
          y_pred

Out[20]: array([1, 1, 1, 1, 1, 6, 1, 1, 1, 1, 4, 6, 6, 2, 7, 1, 1, 2, 4, 1, 4],
               dtype=int64)
```

Result:

Naïve Bayes Classifier algorithm was implemented and the results were noted.

Experiment: 6 **Date:**

Aim: To implement hierarchical clustering with two different datasets.

Procedure:

Data Loading and Inspection:

- Read the iris dataset.

Data Preprocessing:

- Extract relevant features (SepalLengthCm, PetalWidthCm, PetalLengthCm).
- Encode the 'Species' column using LabelEncoder.

Hierarchical Clustering Dendrogram:

- Perform hierarchical clustering on the selected features.
- Visualize the dendrogram

Agglomerative Clustering:

- Perform agglomerative clustering with a specified number of clusters (e.g., 3).
- Visualize the dendrogram for the clustering result.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df=pd.read_csv(r" iris.csv")
df
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
X = df[['SepalLengthCm', 'PetalWidthCm', 'PetalLengthCm']]
label_encoder = LabelEncoder()
```

```
df['encoded_category'] = label_encoder.fit_transform(df['Species'])
y = df['encoded_category']
linkage_matrix = linkage(X, method='ward')
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
n_clusters = 3 # You can choose the number of clusters based on the d
agg_clustering = AgglomerativeClustering(n_clusters=n_clusters)
agg_clustering.fit(X)
linkage_data = linkage(X, method='ward', metric='euclidean')
linkage_data
dendrogram(linkage_data)
plt.show()
```

Output:

Result:

Hierarchical clustering was implemented and results were noted.

Experiment: 7 **Date:**

Aim: To implement K- Mean clustering algorithm.

Procedure:

Data Generation:

- Create a synthetic dataset using **make_blobs** from **sklearn. datasets**. Data Preprocessing:

K – Means Clustering:

- Import the KMeans module from **sklearn.cluster**.
- Create a KMeans model with a specified number of clusters (e.g., 4).
- Fit the model and predict cluster labels for the dataset.

Visualization:

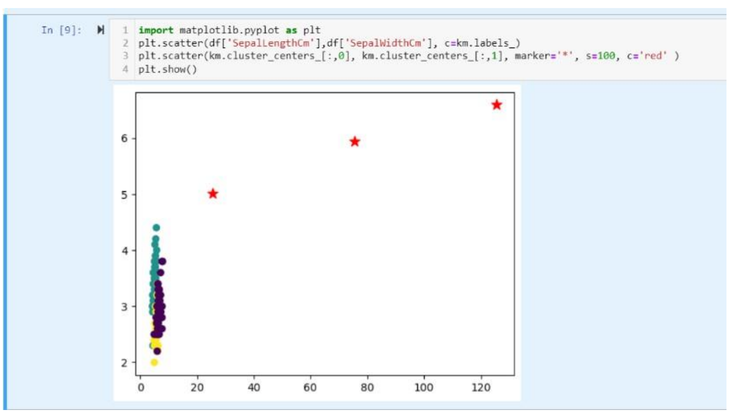
- Create Plot the data points with colors representing the predicted clusters.
- Mark the cluster centers with red asterisks.

Source Code:

```
from sklearn.cluster import KMeans
import sklearn.cluster as KMeans
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# label encoding
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df=pd.read_csv(r" iris.csv")
le.fit(df['Species'])
df['Species']=le.transform(df['Species'])
df.head()
from sklearn.cluster import KMeans
km=KMeans(n_clusters=3)
km.fit(df)
km.fit_predict(df)
import matplotlib.pyplot as plt
```

```
plt.scatter(df['SepalLengthCm'],df['SepalWidthCm'], c=km.labels_)
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], marker='*', s=100, c='red' )
plt.show()
```

Output:



Result:

K-Means Clustering was implemented and results were noted.

Experiment: 8 **Date:**

Aim: Using feature forward selection approaches, reduce the dimensionality of the KDD.

Procedure:

Data Loading:

- Import the dataset and read. [Diabetes Dataset]

Data Preprocessing:

- Extract features (X) and labels (Y) from the dataset. Handle missing values if any.
- Split the dataset into training and testing sets for model evaluation.
- Implement Forward Feature Elimination using SequentialFeatureSelector from mlxtend.
- View the selected features and their corresponding scores.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import mlxtend
import pandas as pd
import numpy as np
df=pd.read_csv(r" diabetes.csv")
df.head()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector
df1=df['Outcome']
df=df.drop('Outcome',axis=1)
x_train, x_test, y_train, y_test = train_test_split(df, df1, test_size=0.2, random_state=42)
lr = LogisticRegression(max_iter=1000)
sfs = SequentialFeatureSelector(lr, k_features=(1, 8), forward=True, verbose=2, cv=5)
sfs = sfs.fit(x_train, y_train)
```

import pandas as pd

pd.DataFrame.from_dict(sfs.get_metric_dict()).T #.T is for transform

sfs.k_feature_names_

sfs.k_score_

Output:

In [13]:
```
import pandas as pd
pd.DataFrame.from_dict(sfs.get_metric_dict()).T #.T is for transform
```

Out[13]:

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
1	(1,)	[0.7073170731707317, 0.7642276422764228, 0.772...	0.745942	(Glucose,)	0.030379	0.023636	0.011818
2	(1, 5)	[0.7479674796747967, 0.7804878048780488, 0.756...	0.758976	(Glucose, BMI)	0.018987	0.014772	0.007386
3	(0, 1, 5)	[0.7560975609756098, 0.8048780487804879, 0.788...	0.775263	(Pregnancies, Glucose, BMI)	0.030485	0.023718	0.011859
4	(0, 1, 5, 6)	[0.7479674796747967, 0.8048780487804879, 0.780...	0.770399	(Pregnancies, Glucose, BMI, DiabetesPedigreeFu...	0.038965	0.030316	0.015158
5	(0, 1, 2, 5, 6)	[0.7560975609756098, 0.8048780487804879, 0.780...	0.775277	(Pregnancies, Glucose, BloodPressure, BMI, Dia...	0.031057	0.024164	0.012082
6	(0, 1, 2, 3, 5, 6)	[0.7560975609756098, 0.7967479674796748, 0.772...	0.768759	(Pregnancies, Glucose, BloodPressure, SkinThic...	0.029634	0.023057	0.011528
7	(0, 1, 2, 3, 5, 6, 7)	[0.7560975609756098, 0.8048780487804879, 0.739...	0.767146	(Pregnancies, Glucose, BloodPressure, SkinThic...	0.035516	0.027632	0.013816
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.7642276422764228, 0.8048780487804879, 0.739...	0.765507	(Pregnancies, Glucose, BloodPressure, SkinThic...	0.03548	0.027605	0.013802

In [14]:
```
sfs.k_feature_names_
```

Out[14]: ('Pregnancies', 'Glucose', 'BloodPressure', 'BMI', 'DiabetesPedigreeFunction')

In [15]:
```
sfs.k_score_
```

Out[15]: 0.7752765560442489

Result:

Forward Feature Elimination with logistic regression was implemented and results were noted.

Experiment: 9 **Date:**

Aim: Using feature backward elimination approaches, reduce the dimensionality.

Procedure:

Data Loading and Inspection:

- Import the dataset using Pandas and inspect the first few rows, data types, and missing values.

Data Preprocessing:

- Extract features (X) and labels (Y) from the dataset. Handle missing values if any.
- Encode categorical labels using Label Encoder.

Data Splitting:

- Split the dataset into training and testing sets for model evaluation.

Linear Regression Model Training and Evaluation:

- Create a linear regression model and train it on the training set.
- Implement Backward Feature Elimination using SequentialFeatureSelector from mlxtend.

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import mlxtend
import pandas as pd
import numpy as np
df=pd.read_csv(r" diabetes.csv")
df.head()
df1=df.pop('Outcome')
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df, df1, test_size = 0.2)
#from sklearn.ensemble import RandomForestClassifier
#rfc=RandomForestClassifier()
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
#cv =  cross validation
```

from mlxtend.feature_selection import SequentialFeatureSelector

sfs1=SequentialFeatureSelector(lr, k_features=(1,8), forward=False, verbose=4, cv=5)

sfs1.fit(x_train, y_train)

pd.DataFrame.from_dict(sfs1.get_metric_dict()).T #.T is for transform

sfs1.k_feature_names_

sfs1.k_score_

Output:

```
Out[5]: SequentialFeatureSelector(estimator=LinearRegression(), forward=False,
                                  k_features=(1, 8), scoring='r2', verbose=4)

In [6]: pd.DataFrame.from_dict(sfs1.get_metric_dict()).T #.T is for transform

Out[6]:
```

	feature_idx	cv_scores	avg_score	feature_names	ci_bound	std_dev	std_err
8	(0, 1, 2, 3, 4, 5, 6, 7)	[0.20901624463622348, 0.3087911216482687, 0.24...	0.278643	(Pregnancies, Glucose, BloodPressure, SkinThic...	0.054754	0.042601	0.0213
7	(0, 1, 2, 4, 5, 6, 7)	[0.21737536816585878, 0.30895531888097927, 0.2...	0.283598	(Pregnancies, Glucose, BloodPressure, Insulin,...	0.053882	0.041922	0.020961
6	(0, 1, 2, 5, 6, 7)	[0.2228971652203401, 0.3114343154764604, 0.244...	0.283168	(Pregnancies, Glucose, BloodPressure, BMI, Dia...	0.052663	0.040973	0.020487
5	(0, 1, 2, 5, 7)	[0.22732537423811172, 0.3087821643969013, 0.23...	0.280087	(Pregnancies, Glucose, BloodPressure, BMI, Age)	0.053772	0.041836	0.020918
4	(0, 1, 2, 5)	[0.23521673996445824, 0.29051266934722964, 0.2...	0.27591	(Pregnancies, Glucose, BloodPressure, BMI)	0.051855	0.040345	0.020172
3	(0, 1, 5)	[0.23352772274668687, 0.2737858461720529, 0.22...	0.273143	(Pregnancies, Glucose, BMI)	0.050812	0.039534	0.019767
2	(1, 5)	[0.2115335520235463, 0.249405064513207, 0.1990...	0.244159	(Glucose, BMI)	0.04799	0.037338	0.018669
1	(1,)	[0.19061102637016603, 0.24191516232035293, 0.1...	0.215243	(Glucose,)	0.059148	0.046019	0.023009

```
In [7]: sfs1.k_feature_names_
Out[7]: ('Pregnancies',
         'Glucose',
         'BloodPressure',
         'Insulin',
         'BMI',
         'DiabetesPedigreeFunction',
         'Age')

In [8]: sfs1.k_score_
Out[8]: 0.28359816505388025
```

Result:

Forward Feature Elimination with logistic regression was implemented and results were noted.

Experiment: 10 **Date:**

Aim: To implement DBSCAN algorithm with two different datasets.

Procedure:

Importing the Required Libraries:

- Import necessary libraries for clustering

Loading the Data:

- Read the dataset and preprocess it by dropping irrelevant columns. [House Dataset]

Preprocessing the Data:

- Scale and normalize the data.

Reducing Dimensionality:

- Apply PCA to reduce data to 2 dimensions for visualization.

Building the Clustering Model:

- Apply DBSCAN clustering.

Visualizing the Clustering:

- Plot the clustering results.

Tuning Model Parameters:

- Adjust DBSCAN parameters.

Visualizing Parameter Tuning:

- Plot the clustering results with tuned parameters.

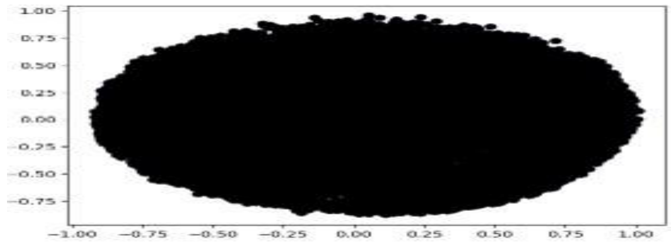
Source Code:

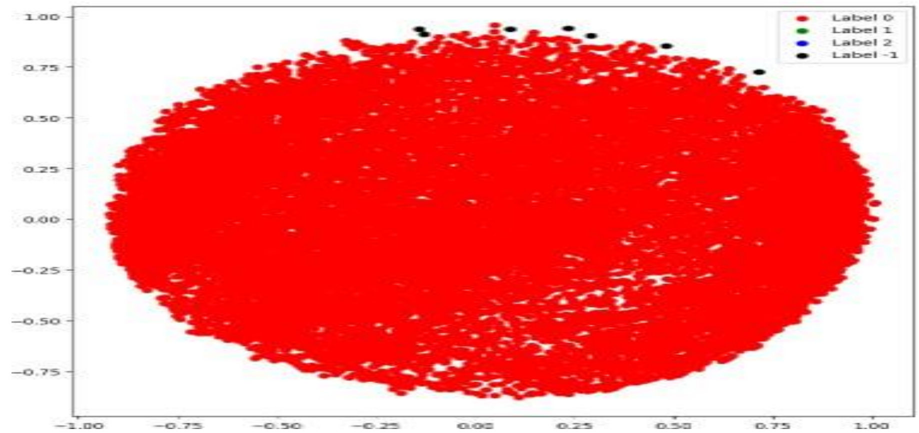
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
X = pd.read_csv(r kc_house_data.csv")
X.drop(['id','date'],axis=1,inplace=True)
```

```
X.head()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_normalized = normalize(X_scaled)
X_normalized = pd.DataFrame(X_normalized)
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
print(X_principal.head())
db_default = DBSCAN(eps = 0.0375, min_samples = 3).fit(X_principal)
labels = db_default.labels_
colours = {}
colours[0] = 'r'
colours[1] = 'g'
colours[2] = 'b'
colours[-1] = 'k'
cvec = [colours[label] for label in labels]

# For the construction of the legend of the plot
r = plt.scatter(X_principal['P1'], X_principal['P2'], color ='r');
g = plt.scatter(X_principal['P1'], X_principal['P2'], color ='g');
b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
k = plt.scatter(X_principal['P1'], X_principal['P2'], color ='k');

# Plotting P1 on the X-Axis and P2 on the Y-Axis
# according to the colour vector defined
plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)

# Building the legend
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1'))

plt.show()
```

```
db = DBSCAN(eps = 0.0375, min_samples = 50).fit(X_principal)
labels1 = db.labels_

colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[3] = 'c'
colours1[4] = 'y'
colours1[5] = 'm'
colours1[-1] = 'k'

cvec = [colours1[label] for label in labels]
colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]

r = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[0])
g = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[1])
b = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[2])
c = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[3])
y = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[4])
m = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[5])
k = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors[6])
plt.figure(figsize =(9, 9))

plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, c, y, m, k),('Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4','Label 5', 'Label
-1'),scatterpoints = 1,loc ='upper left',ncol = 3,fontsize = 8)
plt.show()
```

Output:





Result:

DBSCAN Algorithm was implemented and results were noted.