REVA UNIVERSITY
Bengaluru, India

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

MACHINE LEARNING & APPLICATIONS LAB
B22EF0507

Fifth Semester

AY-2024-25

(Prepared in Aug-2024)

Composed by: Sohara Banu A R

INDEX

1. Lab Objectives:

The objectives of this course are to

- To understand the basic concepts and techniques of Machine Learning through python programming.
- To develop skills of using recent machine learning packages for solving practical problems.
- To gain experience of doing independent study and research.
- To gain good knowledge in Supervised and Unsupervised Algorithms.

2. Lab Outcomes:

On successful completion of this course; student shall be able to:

CO#	Course Outcomes	POs	PSOs
CO1	Make use of predictive data analytics tools to analyse the characteristics of datasets.	1 to 6, 9,10,12	3
CO2	Choose machine learning technique and computing environment suitable for the given application.	1 to 6, 9,10, 12	1
CO3	Apply a linear regression model for the given real world application.	1 to 6, 9,10,12	3
CO4	Develop an application to make use of decision trees to solve the real world problem.	1 to 6, 9,10,12	2,3
CO5	Make use ofunsupervised learning concepts and dimensionality Prediction techniques in real world applications.	1 to 6, 9,10, 12	1,3
CO6	Apply Machine Learning algorithms in real-world applications using Python programming.	1 to 6, 9,10,12	2,3

3. Lab Requirements:

The following are the required hardware and software for this lab.

Hardware Requirements: A standard personal computer or laptop with the following minimum specifications:

1. **Processor**: Any modern multi-core processor (e.g., Intel Core i5 or AMD Ryzen series).
2. **RAM**: At least 4 GB of RAM for basic programs; 8 GB or more is recommended for larger data structures and complex algorithms.
3. **Storage**: A few gigabytes of free disk space for the development environment and program files.
4. **Display**: A monitor with a resolution of 1024x768 or higher is recommended for comfortable coding.
5. **Input Devices**: Keyboard and mouse (or other input devices) for coding and interacting with the development environment.

Software Requirements:

1. **Operating System**: You can develop and execute C programs for Algorithms Lab on various operating systems, including Windows, macOS, and Linux.
2. **Text Editor or Integrated Development Environment (IDE)**:

   - **Text Editor:** You can use a simple text editor like Notepad (Windows), Nano (Linux/macOS), or any code-oriented text editor like Visual Studio Code, Sublime Text, or Atom.
   - **IDE:** Consider using a python integrated development environment like Jupyter, PyCharm, spyder , R Studio for a more feature-rich coding experience.
3. **C Compiler:**
   - To compile and execute python programs, you need a python compiler like Jupyter notebook, PyCharm, Spyder are apopular choice

4. Guidelines to Students:

.

- ➢ Equipment in the lab for the use of the student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage caused is punishable.
- ➢ Students are required to carry their observation / programs book with completed exercises while entering the lab.
- ➢ Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- ➢ The lab can be used in free time / lunch hours by the students who need to use the systems should get prior permission from the lab in-charge.
- ➢ Lab records need to be submitted on or before the date of submission.
- ➢ Students are not supposed to use flash drives.

Practice

Here are some key activities involved in the developing any Machine Learning Model:

- Problem Understanding: The first step is to clearly understand the problem at hand. This involves identifying the objectives and requirements.
- Understand and identify data needs: Once the problem is understood, the next step is to determine what data is necessary to build the model. This involves in identifying type and quantity of data, data sources and locations.
- Data collection: It is a crucial step in the creation of a machine learning model, as it lays the foundation for building accurate models. In this phase of machine learning model development, relevant data is gathered from various sources to train the machine learning model and enable it to make accurate predictions.
- Preprocessing and Preparing Data : It involves transforming raw data into a format that is suitable for training and testing for our models. This phase aims to clean i.e. remove null values, and garbage values, and normalize and preprocess the data to achieve greater accuracy and performance of our machine learning models.
- Selecting the Right Machine Learning Model: Selecting the right machine learning model plays a pivotal role in building of successful model, with the presence of numerous algorithms and techniques available easily, choosing the most suitable model for a given problem significantly impacts the accuracy and performance of the model.

- Training the Model: In training, you pass the prepared data to your machine learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.
- Evaluating the Model: After training your model, you have to check to see how it's performing. This is done by testing the performance of the model on previously unseen data. The unseen data used is the testing set that you split our data into earlier.

These activities are iterative and may involve revisiting earlier stages as the model development progresses. The goal is to create efficient and reliable models that can effectively solve specific problems.

5. List of Lab Exercises:

Sl No.	Title of the Exercise
	Part A
1	Write a program to predict the price of the house using linear regression algorithm. Evaluate the same using Mean Squared Error and Root Mean Squared Error.
2	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
3	Write a program to classify the given instance using logistic regression. Evaluate the model using classification accuracy.
4	Write a program to classify the instances using Naive Bayes algorithm. Evaluate the same using different metrics.
5	Apply support vector machine to either classification / regression dataset of your choice. Analyse the performance of the same using different metrics.
6	Apply any ensemble-based learning algorithm to any dataset of your choice. Check the performance of this algorithm with other ML algorithms.
7	Write a program to analyse the nearest instances to the given instances using K Nearest Neighbor algorithm. Evaluate the performance of the algorithm using different metrics.
8	Write a program to cluster the instances for unlabelled dataset using K Means clustering algorithm.
9	Write a program to reduce the dimension of the dataset using PCA.
10	Write a program to find the most specific hypothesis using Find-S algorithm. Read the training data from a .CSV file.

LAB EXERCISES

6. Lab Exercises' Solutions:

Program 1 : Write a program to predict the price of the house using linear regression algorithm. Evaluate the same using Mean Squared Error and Root Mean Squared Error.

Problem Statement: Predicting House Prices Using Linear Regression

Solution Overview: The solution involves collecting and preprocessing house price data, selecting relevant features, and implementing a linear regression model using Python. The model is trained to predict house prices based on these features. Performance is evaluated using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to assess prediction accuracy. Finally, the results are analyzed to understand the model's effectiveness in predicting house prices.

Intuition: The price of a house is influenced by various factors such as the size of the house, the number of bedrooms and bathrooms, the location, and other features. These factors, known as independent variables, collectively determine the dependent variable, which in this case is the price of the house. Linear regression is a statistical method that models the relationship between the dependent variable and one or more independent variables by fitting a linear equation to the observed data.

The idea behind linear regression is to find the best-fitting straight line (in the case of simple linear regression) or hyperplane (in the case of multiple linear regression) that minimizes the difference between the actual and predicted values of the dependent variable. This difference is often measured using the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), which provide insights into the accuracy of the model.

Steps in the Solution:

1. **Data Collection:**

- Gather a dataset containing historical data on house prices along with relevant features such as square footage, number of rooms, location, etc.

2. **Data Preprocessing:**

- Handle any missing values in the dataset, as these can affect the accuracy of the model.
- Encode categorical variables (e.g., location) using techniques such as one-hot encoding.
- Normalize or standardize numerical features to ensure that all features contribute equally to the model.

3. **Feature Selection:**

- Identify the most significant features that influence house prices. This step might involve statistical tests or techniques like correlation analysis.

4. **Model Implementation:**

- Use Python and libraries like scikit-learn to implement a linear regression model.
- Split the data into training and testing sets to evaluate the model's performance.

5. **Model Training:**

- Train the linear regression model on the training dataset. The model will learn the relationship between the features and the house prices.

6. **Model Prediction:**

- Use the trained model to predict house prices on the testing dataset.

7. **Model Evaluation:**

- Evaluate the model's performance using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).
- MSE measures the average of the squares of the errors—that is, the average squared difference between the actual and predicted values.
- RMSE is the square root of MSE and provides an error metric in the same units as the target variable (house prices).

8. **Interpretation:**

- Analyse the results to understand how well the model is performing. A lower MSE and RMSE indicate a better-fitting model.

Code Implementation:

Dataset url: https://data.world/swarnapuri-sude/house-data

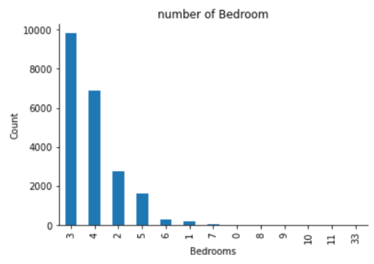
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv("kc_house_data.csv")
data.head()
```

```
data  = data.drop(["id", "date"], axis = 1)
data.head()
data.describe()
data['bedrooms'].value_counts().plot(kind='bar')
plt.title('number of Bedroom')
plt.xlabel('Bedrooms')
plt.ylabel('Count')
sns.despine()
plt.figure(figsize=(10,10))
sns.jointplot(x=data.lat.values, y=data.long.values, height=10)
plt.ylabel('Longitude',fontsize=12)
plt.xlabel('Latitude',fontsize=12)
sns.despine()
plt.show()
plt.scatter(data.price,data.sqft_living)
plt.title("Price vs Square Feet")
plt.scatter(data.price,data.long)
plt.title("Price vs Location of the area")
plt.scatter(data.price,data.lat)
plt.xlabel("Price")
plt.ylabel('Latitude')
plt.title("Latitude vs Price")
plt.scatter(data.bedrooms,data.price)
plt.title("Bedroom and Price ")
plt.xlabel("Bedrooms")
plt.ylabel("Price")
sns.despine()
plt.show()
plt.scatter((data['sqft_living']+data['sqft_basement']),data['price'])
plt.scatter(data.waterfront,data.price)
plt.title("Waterfront vs Price ( 0= no waterfront)")
#Extracting X features and y label
y = data['price']
```

```
X = data.drop(['price'],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 = 0.10,random_state =2)
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
reg.score(x_test,y_test)
```

Sample Output:

Sample Visualization:



Result = 0.7320342760357638

Outcome of the Exercise:

The outcome of the exercise is a trained linear regression model that predicts house prices based on selected features. The model's accuracy is quantified using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). A lower MSE and RMSE indicate better predictive performance. The results provide insights into how well the model generalizes to unseen data. This exercise demonstrates the application of linear regression for real-world price prediction tasks.

Viva/Interview Questions:

Question: What is the purpose of this exercise?

Answer: The purpose of this exercise is to build a linear regression model that predicts house prices based on various features and to evaluate the model's accuracy using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).

Question: What is linear regression?

Answer: Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

Question: Why did you choose linear regression for this problem?

Answer: Linear regression is a simple and interpretable model that is effective for predicting continuous variables, like house prices, especially when the relationship between the features and the target variable is approximately linear.

Question: What are Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)?

Answer: MSE measures the average squared difference between the actual and predicted values, while RMSE is the square root of MSE, providing an error metric in the same units as the target variable.

Question: Why is it important to split the data into training and testing sets?

Answer: Splitting the data ensures that the model is trained on one portion of the data and tested on another, allowing us to evaluate its performance on unseen data and avoid overfitting.

Question: What steps did you take to preprocess the data?

Answer: Data preprocessing involved handling missing values, encoding categorical variables, normalizing numerical features, and selecting the most relevant features for the model.

Question: How did you select the features for the model?

Answer: Feature selection was done by analyzing the correlation between the features and the target variable, and by considering domain knowledge to identify the most significant factors influencing house prices.

Question: What is the significance of a low MSE and RMSE?

Answer: A low MSE and RMSE indicate that the model's predictions are close to the actual values, meaning the model has good predictive accuracy and generalizes well to new data.

Question: How would you improve the model if the RMSE is high?

Answer: To improve the model, I would consider adding more relevant features, using more advanced techniques like regularization to reduce overfitting, or trying different types of models such as polynomial regression or decision trees.

Question: Can linear regression capture non-linear relationships between features and the target variable?

Answer: Linear regression assumes a linear relationship, so it may not capture non-linear relationships effectively. However, non-linear relationships can sometimes be modeled by transforming the features (e.g., polynomial features) before applying linear regression.

Program 2: 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

Problem Statement: Design and implement a program to demonstrate the working of the ID3 (Iterative Dichotomiser 3) decision tree algorithm. Use a dataset to build a decision tree that can classify new data samples. The program should illustrate the process of selecting attributes, splitting nodes, and making decisions based on entropy and information gain. Test the decision tree by classifying a new sample and evaluate its performance.

Solution Overview: The ID3 algorithm builds a decision tree by recursively splitting the dataset based on the attribute that provides the maximum information gain. The dataset is divided into subsets that are as pure as possible, meaning they contain samples of the same class. The process continues until the algorithm can classify the samples perfectly or no further splitting is possible. A new sample is then classified by traversing the decision tree from the root to a leaf node.

Intuition: The core idea behind the ID3 algorithm is to select the attribute that best separates the data into classes at each step. This is measured using information gain, which is calculated based on the reduction in entropy after splitting the dataset. By choosing the attribute with the highest information gain at each node, the decision tree becomes an efficient classifier that mimics human decision-making processes.

Steps in the Solution:

Load the dataset: Import a dataset that contains features and corresponding class labels.

Calculate Entropy and Information Gain: For each attribute, compute the entropy and information gain.

Build the Decision Tree: Recursively split the dataset based on the attribute with the highest information gain.

Classify New Sample: Use the constructed decision tree to classify a new, unseen sample.

Evaluate the Tree: Assess the accuracy and effectiveness of the decision tree by comparing its predictions with actual outcomes.

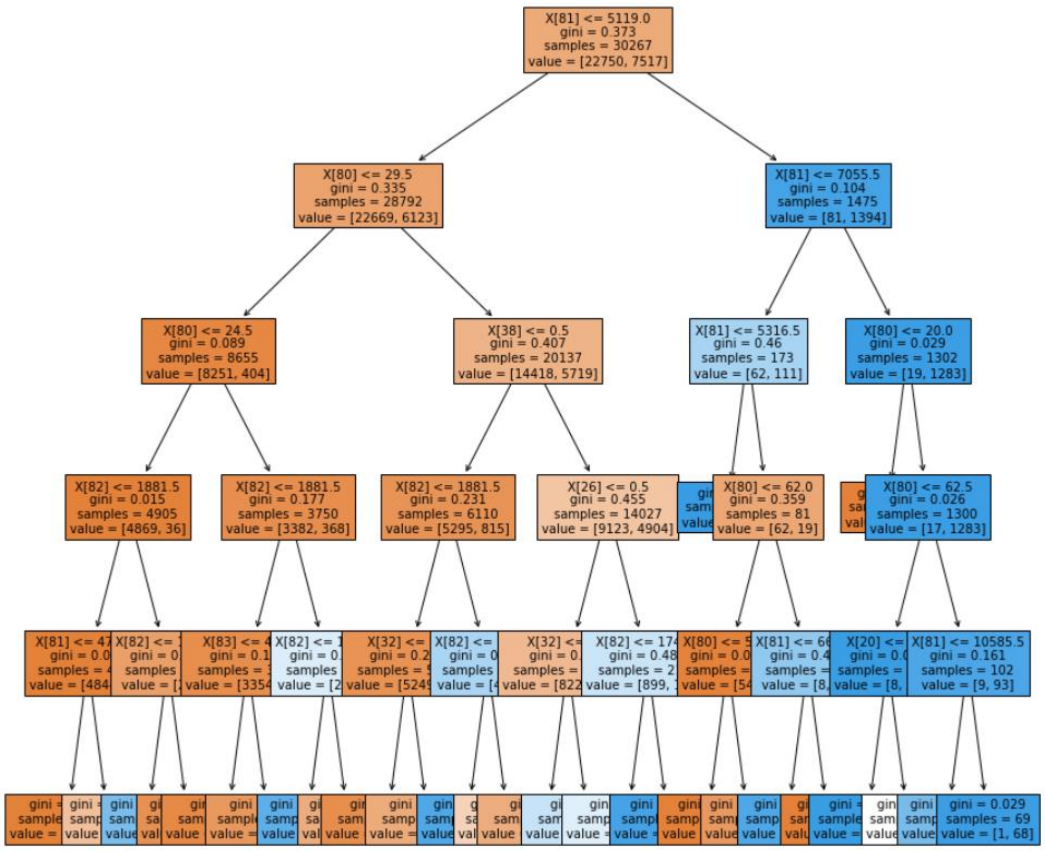
Code Implementation:

Dataset: The Adult dataset is from the Census Bureau and the task is to predict whether a given adult earns more than $50,000 a year or not based attributes such as education, hours of work per week, etc.. URL: https://www.kaggle.com/datasets/wenruliu/adult-income-

```
#Required imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#Read dataset
df = pd.read_csv("adult.csv")
df.head()
df.columns
df.shape
# See the columns that contain a "?" and how many "?" are there in those columns
df.isin(['?']).sum()
#Replace ? with NaN
df['workclass'] = df['workclass'].replace('?', np.nan)
df['occupation'] = df['occupation'].replace('?', np.nan)
df['native-country'] = df['native-country'].replace('?', np.nan)
#Now the ? has been replaced by NaN, so count of ? is 0
df.isin(['?']).sum()
#Check missing values - NaN values
df.isnull().sum()
#Drop all rows that contain a missing value
df.dropna(how='any', inplace=True)
#Check duplicate values in dataframe now
print(f"There are {df.duplicated().sum()} duplicate values")
df = df.drop_duplicates()
df.shape
df.columns
#Drop non-relevant columns
df.drop(['fnlwgt','educational-num','marital-status','relationship', 'race',], axis = 1, inplace = True)
df.columns
#Extract X and y from the dataframe , income column is the target column, rest columns are features
X = df.loc[:,['age', 'workclass', 'education', 'occupation', 'gender', 'capital-gain',
        'capital-loss', 'hours-per-week', 'native-country']]
y = df.loc[:,'income']
X.head()
y.head()
# Since y is a binary categorical column we will use label encoder to convert it into numerical columns with values 0 and 1
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()
#First identify caterogical features and numeric features
numeric_features = X.select_dtypes('number')
categorical_features = X.select_dtypes('object')
categorical_features
numeric_features
#Convert categorical features into numeric
converted_categorical_features = pd.get_dummies(categorical_features)
converted_categorical_features.shape
#combine the converted categorical features and the numeric features together into a new dataframe called "newX"
all_features = [converted_categorical_features, numeric_features]
newX = pd.concat(all_features,axis=1, join='inner')
newX.shape
newX.columns
#Do a train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(newX, y, test_size=0.33, random_state=42)
# Load Decision Tree Classifier, max_depth = 5 and fit it with X-train and y-train
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(X_train, y_train)
# Load Decision Tree Classifier, max_depth = 5 and fit it with X-train and y-train
from sklearn.tree import DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(X_train, y_train)
# Make predictions
y_pred = clf.predict(X_test)
y_test.shape
y_pred.shape
predictions_df = pd.DataFrame()
predictions_df['precdicted_salary_class'] = y_pred
predictions_df['actual_salary_class'] = y_test[0].values
predictions_df
#Evaluate the performance of fitting
from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred,y_test))
#Plot your decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
plt.figure(figsize=(14,14))
plot_tree(clf, fontsize=10, filled=True)
plt.title("Decision tree trained on the selected features")
plt.show()
```

Sample Output:



Outcome of the Exercise:

The exercise results in a decision tree capable of classifying new samples based on the patterns learned from the dataset. The tree visually and computationally demonstrates the importance of attributes in decision-making. By understanding the working of the ID3 algorithm, students gain insights into how decision trees operate and how to apply them to real-world problems.

Viva/Interview Questions:

What is the ID3 algorithm?

The ID3 algorithm is a decision tree algorithm that uses entropy and information gain to build a tree for classifying data samples.

How does ID3 select the best attribute for splitting the data?

ID3 selects the attribute with the highest information gain, which measures the reduction in entropy after the split.

What is entropy in the context of the ID3 algorithm?

Entropy is a measure of the impurity or randomness in the dataset. It indicates how mixed the classes are in a subset.

What is information gain?

Information gain is the reduction in entropy after a dataset is split on an attribute. It quantifies how well an attribute separates the data into classes.

Can the ID3 algorithm handle continuous data?

ID3 is designed for categorical data, but it can handle continuous data by discretizing it into intervals.

What are the limitations of the ID3 algorithm?

ID3 tends to overfit the training data, especially when the tree becomes too complex. It also struggles with continuous data and missing values.

How does ID3 differ from the C4.5 algorithm?

C4.5 is an extension of ID3 that can handle continuous data, prune trees to avoid overfitting, and deal with missing values.

Why is it important to avoid overfitting in decision trees?

Overfitting occurs when the model is too complex and captures noise in the training data, leading to poor performance on new, unseen data.

What is a leaf node in a decision tree?

A leaf node is a terminal node in the decision tree that represents a class label and where no further splitting is possible.

How can the performance of a decision tree be evaluated?

The performance of a decision tree can be evaluated using metrics like accuracy, precision, recall, and F1-score on a test dataset.

Program 3: Write a program to classify the given instance using logistic regression. Evaluate the model using classification accuracy.

Problem Statement: to classify the given instance and evaluate

Solution Overview: A statistical model for binary classification is called logistic regression. Using the sigmoid function, it forecasts the likelihood that an instance will belong to a particular class, guaranteeing results between 0 and 1. To minimize the log loss, the model computes a linear combination of input characteristics, transforms it using the sigmoid, and then optimizes its coefficients using methods like gradient descent. These coefficients establish the decision boundary that divides the classes. Because of its ease of use, interpretability, and versatility across multiple domains, Logistic Regression is widely used in machine learning for problems that involve binary outcomes. Overfitting can be avoided by implementing regularization.

Intuition: Suppose you want to predict whether a student will pass an exam (Pass/Fail) based on the number of hours they study. Logistic regression would:

- Calculate a weighted sum of the input (hours studied).
- Apply the sigmoid function to get a probability.
- If the probability is above 0.5, predict "Pass"; otherwise, predict "Fail".

Steps in the Solution:

.

Code Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve,
auc
```

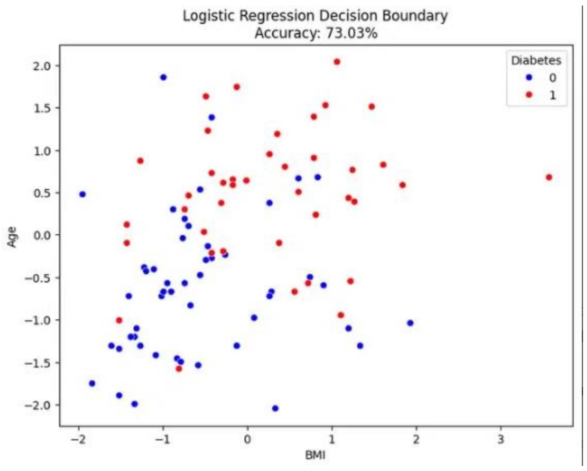
Read and Explore the data

```
 # Load the diabetes dataset
diabetes = load_diabetes()
```

```
X, y = diabetes.data, diabetes.target
 # Convert the target variable to binary (1 for diabetes, 0 for no diabetes)
y_binary = (y > np.median(y)).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y_binary, test_size=0.2, random_state=42)
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualize the decision boundary with accuracy information
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test[:, 2], y=X_test[:, 8], hue=y_test, palette={
                0: 'blue', 1: 'red'}, marker='o')
plt.xlabel("BMI")
plt.ylabel("Age")
plt.title("Logistic Regression Decision Boundary\nAccuracy: {:.2f}%".format(
    accuracy * 100))
plt.legend(title="Diabetes", loc="upper right")
plt.show()
```

Sample Output:

```
Accuracy: 73.03%
Confusion Matrix:
 [[36 13]
 [11 29]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        49
           1       0.69      0.72      0.71        40

    accuracy                           0.73        89
   macro avg       0.73      0.73      0.73        89
weighted avg       0.73      0.73      0.73        89
```



Viva Questions

Q1. What is Logistic Regression?

A statistical technique for binary classification issues is called logistic regression.It uses a logistic function to model the likelihood of a binary outcome occurring.

Q2. How is Logistic Regression different from Linear Regression?

The probability of a binary event is predicted by logistic regression, whereas a continuous outcome is predicted by linear regression. In order to limit the output between 0 and 1, logistic regression uses the logistic (sigmoid) function.

Q3. How to handle categorical variables in Logistic Regression?

Use one-hot encoding, for instance, to transform categorical information into numerical representation. Make sure the data has been properly preprocessed to prepare it for logistic regression.

Q4. Can Logistic Regression handle multiclass classification?

It is possible to use methods like One-vs-Rest or Softmax Regression to expand logistic regression for multiclass classification.

Q5. What is the role of the sigmoid function in Logistic Regression?

Any real integer can be mapped to the range [0, 1] using the sigmoid function. The linear equation’s output is converted into probabilities by it.

Program 4: Write a program to classify the instances using Naive Bayes algorithm. Evaluate the same using different metrics.

Problem Statement: to classify the instances

Solution Overview: This model predicts the probability of an instance belongs to a class with a given set of feature value. It is a probabilistic classifier. It is because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other. In real world, this condition satisfies rarely. It uses Bayes theorem in the algorithm for training and prediction

Intuition: Before looking at the features, the model has a "prior" belief about the likelihood of each class based on historical data.

Steps in the Solution:

Step 1: Calculate the prior probability for given class labels

Step 2: Find Likelihood probability with each attribute for each class

Step 3: Put these value in Bayes Formula and calculate posterior probability.

Step 4: See which class has a higher probability, given the input belongs to the higher probability class.

Code Implementation:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import pandas as pd
dataset = pd.read_csv('iris_data.csv')
dataset.head()
%matplotlib inline
img=mpimg.imread('iris_types.jpg')
plt.figure(figsize=(20,40))
plt.axis('off')
plt.imshow(img)
#Spliting the dataset in independent and dependent variables
X = dataset.iloc[:,:4].values
y = dataset['species'].values
# Splitting the dataset into the Training set and Test set
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 82)
# Feature Scaling to bring the variable in a single scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Fitting Naive Bayes Classification to the Training set with linear kernel
from sklearn.naive_bayes import GaussianNB
nvclassifier = GaussianNB()
nvclassifier.fit(X_train, y_train)
# Predicting the Test set results
y_pred = nvclassifier.predict(X_test)
print(y_pred)
#lets see the actual and predicted value side by side
y_compare = np.vstack((y_test,y_pred)).T
#actual value on the left side and predicted value on the right hand side
#printing the top 5 values
y_compare[:5,:]
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
#finding accuracy from the confusion matrix.
a = cm.shape
corrPred = 0
falsePred = 0

for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cm[row,c]
        else:
```

```
        falsePred += cm[row,c]
print('Correct predictions: ', corrPred)
print('False predictions', falsePred)
print ('\n\nAccuracy of the Naive Bayes Clasification is: ', corrPred/(cm.sum()))
```

Sample Output:

```
Correct predictions:  28
False predictions 2
Accuracy of the Naive Bayes Classification is:  0.9333333333333333
```

Outcome of the Exercise:

The Naive Bayes model has correctly classify a significant portion of the test data based on the features provided. we see the model performs on a given dataset with 93% accuracy.

Viva/Interview Questions:

What mathematical concept Naive Bayes is based on?

Naive Bayes is based on Bayes theorem in statistics. It calculates probabilities independently for each class based on conditions and without conditions and then predicts outcomes based on that.

What are the different types of Naive Bayes classifiers?

MultinomialNaiveBayes

BernoulliNaiveBayes

Gaussian Naive Bayes

Is Naive Bias a classification algorithm or regression algorithm?

It is a classification algorithm. Naive Bayes is a supervised learning algorithm but it can also be trained as semi-supervised learning algorithm.

Program 5: Apply support vector machine to either classification / regression dataset of your choice. Analyse the performance of the same using different metrics.

Problem Statement: Develop SVM for classification or regression. Also evaluate the model

Solution Overview: Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Intuition:

The core idea behind SVM is to find a way to separate data points that belong to different classes as clearly as possible. Imagine you have data points on a plane, and each point belongs to one of two classes (e.g., red or blue). SVM tries to draw a line (in 2D) or a hyperplane (in higher dimensions) that divides these classes with the greatest margin.

Steps in the Solution:

The process includes training, testing and evaluating the model on the Adult dataset. In this experiment you need to train a classifier on the Adult dataset, to predict whether an individual's income is greater or less than $50,000.

Dataset: We have used a smaller version of adult income dataset. This dataset has 3574 rows and 7 columns. It has a total of 15 columns, Target Column is "Income", The income is divide into two classes: <=50K and >50K Number of attributes: 6, These are the demographics and other features to describe a person

6 attributes are:

Age.

Workclass.

Education Number of Years.

Occupation.

gender.

Hours-per-week.

The dataset contains missing values that are marked with a question mark character (?). There are two class values '>50K' and '<=50K' in target column i.e., it is a binary classification task.

Code Implementation:

```
#Required imports

import numpy as np
import pandas as pd
#Read dataset
df = pd.read_csv("smaller_adult.csv")
df.head()
df.columns
df.shape(3574, 7)
df.info()
df.describe()
# See the columns that contain a "?" and how many "?" are there in those columns
df.isin(['?']).sum()
df.columns
#Replace ? with NaN
df['workclass'] = df['workclass'].replace('?', np.nan)
df['occupation'] = df['occupation'].replace('?', np.nan)
#Now the ? has been replaced by NaN, so count of ? is 0
df.isin(['?']).sum()
#Check missing values - NaN values
df.isnull().sum()
#Drop all rows that contain a missing value
df.dropna(how='any', inplace=True)
#Check duplicate values in dataframe now
print(f"There are {df.duplicated().sum()} duplicate values")
df = df.drop_duplicates()
df.shape
df.columns
#Extract X and y from the dataframe , income column is the target column, rest columns are feat
```

```
ures
X = df.loc[:,['age', 'workclass', 'educational-num', 'occupation', 'gender', 'hours-per-week']]
y = df.loc[:,'income']
# Since y is a binary categorical column we will use label encoder to convert it into numerical co
lumns with values 0 and 1
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()
#First identify caterogical features and numeric features
numeric_features = X.select_dtypes('number')
categorical_features = X.select_dtypes('object')

#Convert categorical features into numeric
converted_categorical_features = pd.get_dummies(categorical_features)
converted_categorical_features.shape
#combine the converted categorical features and the numeric features together into a new datafra
me called "newX"

all_features = [converted_categorical_features, numeric_features]
newX = pd.concat(all_features,axis=1, join='inner')
newX.shape
newX.columns
#Do a train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(newX, y, test_size=0.33, random_state=42)
# Load Support Vector Machine Classifier
from sklearn.svm import SVC
clf = SVC(kernel="linear", gamma = 'auto')
clf.fit(X_train, y_train.values.ravel())
SVC(gamma='auto', kernel='linear')
# Make predictions
y_pred = clf.predict(X_test)
```

```
predictions_df = pd.DataFrame()
predictions_df['precdicted_salary_class'] = y_pred
predictions_df['actual_salary_class'] = y_test[0].values
predictions_df
#Evaluate the performance of fitting
from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred,y_test))
```

Output

```
0.7672162948593598
```

Outcome of the program

This output shows the classifier name along with the accuracy score on the test data for each classifier. The scores indicate how well each classifier performed

Viva Questions

What is a Support Vector Machine?

Answer: A Support Vector Machine (SVM) is a supervised machine learning algorithm that is used for classification and regression tasks. It finds the hyperplane that best separates the data into different classes. In the case of two-dimensional data, this hyperplane is a line, but in higher dimensions, it becomes a plane or a hyperplane.

What is a hyperplane in the context of SVM?

Answer: A hyperplane is a decision boundary that separates different classes in the feature space. In an n-dimensional space, the hyperplane is an (n-1)-dimensional subspace that divides the space into two halves, each corresponding to a different class.

What are support vectors?

Answer: Support vectors are the data points that are closest to the hyperplane and influence its position and orientation. They are critical to the construction of the hyperplane as they define the margin of separation between the classes.

Explain the concept of the margin in SVM.

Answer: The margin in SVM refers to the distance between the hyperplane and the nearest data points from either class. SVM aims to maximize this margin to ensure that the classifier has the best possible generalization capability on unseen data.

What is the difference between a hard margin and a soft margin in SVM?

Answer: A hard margin SVM requires that all data points are correctly classified, meaning no points are allowed inside the margin. A soft margin SVM, on the other hand, allows some misclassification or points within the margin, which is useful in cases where the data is not linearly separable.

How does SVM handle non-linearly separable data?

Answer: SVM handles non-linear data by using a technique called the kernel trick. Kernels map the original features into a higher-dimensional space where a linear separator (hyperplane) can be found. Common kernels include the polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel.

What is the kernel trick?

Answer: The kernel trick is a mathematical technique that allows SVMs to operate in a high-dimensional feature space without explicitly computing the coordinates of the data in that space. It computes the dot product of the data points in the feature space, making it computationally feasible to find a separating hyperplane in cases where the data is not linearly separable in the original space.

What are the common types of kernels used in SVM?

Answer: The common types of kernels used in SVM include:

Linear Kernel: Suitable for linearly separable data.

Polynomial Kernel: Good for data where the relationship between features is polynomial.

Radial Basis Function (RBF) Kernel: Effective when there is a need to handle non-linear relationships.

Sigmoid Kernel: Often used as a proxy for neural networks.

Program 6: Apply any ensemble-based learning algorithm to any dataset of your choice. Check the performance of this algorithm with other ML algorithms.

Problem statement: Apply any ensemble-based learning algorithm to any dataset of your choice. Check the performance of this algorithm with other ML algorithms.

Solution Overview:

1. Importing Libraries

The code begins by importing essential libraries:

NumPy and Pandas are used for data manipulation and analysis.

Scikit-learn provides various machine learning algorithms and utilities for model evaluation and preprocessing.

2. Defining Classifiers

The code defines a list of classifier names and their corresponding instances:

Each classifier is initialized with specific parameters that can be tuned for better performance.

3. Loading and Preprocessing Data

The dataset is loaded, and unnecessary columns are dropped:

The Iris dataset is a well-known dataset used for classification tasks, containing features like sepal length, sepal width, petal length, and petal width.

4. Feature and Target Extraction

The features (X) and target (y) are extracted from the DataFrame:

5. Encoding Categorical Target Variable

Since the target variable is categorical, it is transformed into numerical format using LabelEncoder:

6. Train-Test Split

The dataset is split into training and testing sets:

train_test_split function is used to create a training set (67%) and a testing set (33%).

7. Training Classifiers and Evaluating Performance

A for loop is used to train each classifier and evaluate its accuracy:

Each classifier is fitted to the training data, and the accuracy score is calculated on the test data.

The results are printed, showing the performance of each classifier

Intuition

The overall intuition behind the code is to create a systematic approach to machine learning classification. By leveraging multiple classifiers, the code aims to identify which algorithm performs best for the Iris dataset. This process of experimentation and evaluation is a fundamental aspect of developing effective machine learning models. The insights gained from this code can be applied to other datasets and classification tasks, making it a versatile framework for understanding machine learning workflows.

Code Implementation

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

names = [
    "K-Nearest Neighbors",
    "Linear SVM",
    "Decision Tree",
    "Multilayer Perceptron",
    "Gaussian Naive Bayes",
    "Random Forest"
]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    DecisionTreeClassifier(max_depth=5),
    MLPClassifier(alpha=1, max_iter=1000),
    GaussianNB(),
    RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
]
```

```
df = pd.read_csv("Iris.csv")
df.head()
df = df.drop("Id", axis=1)
df.head()

# Extract X and y as features and target
X = df.iloc[:, :-1]
X.head()
y = df.iloc[:, -1]
y.head()

# Since target column is categorical, we will convert it to numerical using LabelEncoder
from sklearn.preprocessing import LabelEncoder
y = LabelEncoder().fit_transform(y)
y = pd.DataFrame(y)
y.head()

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Using a for loop fit all the classifiers to X_train and y_train and print the accuracy score of
each classifier
for name, clf in zip(names, classifiers):
   clf.fit(X_train, y_train.values.ravel())
   score = clf.score(X_test, y_test)
   print("Classifier Name: ", name, "Score: ", score)
```

Sample Output

```
Classifier Name:  K-Nearest Neighbors Score:  0.98
Classifier Name:  Linear SVM Score:  0.96
Classifier Name:  Decision Tree Score:  0.98
Classifier Name:  Multilayer Perceptron Score:  0.98
Classifier Name:  Gaussian Naive Bayes Score:  0.96
```

Classifier Name: Random Forest Score: 0.98

Outcome of the program

This output shows the classifier name along with the accuracy score on the test data for each classifier. The scores indicate how well each classifier performed on the Iris dataset.

Viva/Interview Questions

What is the purpose of this code?

The code aims to classify Iris flower species using various machine learning algorithms.

What libraries are being used in this code?

The code uses Pandas for data manipulation, Scikit-learn for machine learning algorithms, and train_test_split for splitting the dataset.

What is the role of the names and classifiers lists?

The names list contains the names of the classifiers, while the classifiers list contains the actual instances of the classifiers with their respective parameters.

Explain the purpose of the train_test_split function.

The train_test_split function is used to divide the dataset into training and testing sets, allowing for model evaluation on unseen data.

How does the code handle categorical target variables?

The code uses LabelEncoder to convert the categorical target variable into numerical format, enabling the classifiers to process the data.

What is the purpose of the for loop in the code?

The for loop iterates over each classifier, fits it to the training data, and evaluates its accuracy on the test data.

What are the different classifiers used in this code?

The code uses K-Nearest Neighbors, Linear SVM, Decision Tree, Multilayer Perceptron, and Gaussian Naive Bayes, Random Forest classifiers.

How can the performance of the classifiers be compared?

The accuracy scores printed for each classifier can be compared to determine which model performs best on the Iris dataset.

What is the significance of the random_state parameter in train_test_split?

The random_state parameter ensures reproducibility of the train-test split, allowing for consistent results across different runs of the code.

Program 7: Write a python program to classify the medical dataset using K Nearest Neighbor Algorithm.

Problem Statement: Design KNN classifier

Solution Overview:

The students are expected to demonstrate how you can perform basic data processing operations, split the dataset into training and test sets, train the model, score the test dataset, and evaluate the predictions.

Intuition:

It simply calculates the distance between a sample data point and all other training data points.. The distance can be Euclidean distance or Manhattan distance. Then, it selects the k nearest data points where k can be any integer.

Steps in the Solution:

Step 1: Selecting the optimal value of K.

Step 2: Calculating distance.

Step 3: Finding Nearest Neighbors.

Step 4: Voting for Classification or Taking Average for Regression.

Code Implementation:

```
# import required libraries import numpy as np import pandas as pd

#if you have downloaded the dataset to you local computer you can use this syntax to read the fileinto
your pandas dataframe

data = pd.read_csv("breast-cancer-wisconsin-data_data.csv")
data.head()
data.columns
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_se',
'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
'fractal_dimension_se', 'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst',
'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
dtype='object')
data = data.drop(['id', 'Unnamed: 32'], axis = 1) data.shape
```

```
data.describe()
data.info()
data.columns

X = data.loc[:, ['radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_se',
'smoothness_se', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
'fractal_dimension_se', 'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst',
'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst']]

y = data.loc[:, 'diagnosis']
#Train - Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
# Fit the KNN Classifier
from sklearn.neighbors import KNeighborsClassifier knn_cfr =
KNeighborsClassifier(n_neighbors=3) knn_cfr.fit(X_train, y_train)
KNeighborsClassifier(n_neighbors=3)
# Use the fitted model to make prediction for test data
y_pred = knn_cfr.predict(X_test)
```

Sample Output:

```
# Print the accuracy score of your model
# accuracy = Number of correct predictions / total number of predictions
# accuracy_score = the number of test samples for which y_pred == y_test

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
0.9414893617021277
# This model has an accuracy of 94.14 %
```

Outcome of the Exercise:

The closer two given points are to each other, the more related and similar they are. Typically used with data transmitted over computer networks.

Viva/Interview Questions:

K Nearest Neighbor algorithm?

k-NN is the simplest supervised learning algorithm. It assumes the similarity between the new data and available cases and puts new data into the category that is most similar. It stores all the available data and classifies a new data point based on similarity. k-NN can also be used for regression problems, but it is mostly used for classification problems. It is also known as a "lazy learner" because it does not learn from the training set immediately. But at the time of classification, it performs the action.

Why is the odd value of 'k' preferred over an even value in the k-NN algorithm?

Using an odd value of 'k' is preferred over an even value because it can prevent ties in the class prediction where k is even, and there is a possibility of having an equal number of neighbours for each class, resulting in a tie.

How does the k-NN algorithm make predictions on unseen datasets?

In k-NN, to make a prediction for a new, unseen data point, the algorithm identifies the k-closest points (i.e., the k-nearest neighbour) in the training dataset based on some distance metrics, and then assigns the class label of the majority of these k neighbours to the new data point.

Why is it recommended not to use the k-NN Algorithm for large datasets?

k-NN works well with small datasets because it is a lazy learner. It needs to store all the data and then make a decision only at runtime. The k-NN algorithm can be computationally expensive, especially for large datasets. This is because the algorithm needs to compute the distance between each pair of data points in the dataset to identify the k-nearest neighbours. As the number of data points increases, the number of distance computations required grows quickly, and this can make the algorithm very slow.

Program 8 : Write a program to cluster the instances for unlabelled dataset using K Means clustering algorithm.

Problem Statement: Given an unlabeled dataset, the task is to cluster the instances into distinct groups based on their features using the K-Means clustering algorithm. The objective is to partition the data into k clusters where each instance is assigned to the cluster with the nearest mean (centroid). The number of clusters k should be determined based on the dataset characteristics. The result should help in understanding the inherent groupings within the data. The output should include cluster assignments and visualizations.

Solution Overview: The solution involves implementing the K-Means clustering algorithm to group the data into kkk clusters. We start by preprocessing the data, including standardizing it for better results. The K-Means algorithm is then applied, initializing centroids, assigning data points to clusters, and updating centroids iteratively. The results are visualized using scatter plots to show cluster distribution and centroids. Additionally, metrics such as inertia and silhouette scores are used to evaluate clustering quality.

Intution:

K-Means clustering works by iteratively assigning data points to the nearest centroid and updating the centroids based on the mean of the assigned points. The algorithm seeks to minimize the variance within each cluster, making clusters as compact and distinct as possible. Choosing the number of clusters kkk is crucial and can be guided by methods like the Elbow Method. Visualization helps in understanding the clustering result, while evaluation metrics ensure that clusters are meaningful and well-separated.

Data Set: wine_clustering.csv

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The attributes are:

- Alcohol
- Malic acid
- Ash

- Alcalinity of ash
- Magnesium
- Total phenols
- Flavanoids
- Nonflavanoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

Steps involved:

1. First determines the number of groups/clusters (called as K),
2. Then it randomly chooses initial K centroids from data points,
3. Next it assigns data points to the cluster of nearest centroid
3. Then it updates centroids in each iteration until clusters converge.

The K in K-Means comes from the number of clusters that need to be set prior to starting the iteration process.

We can choose the best value of K using The Elbow Method.

The best value of K is one that results in groups with minimum variance within a single cluster.

This measure is called Within Cluster Sum of Squares, or WCSS for short. The smaller the WCSS is, the closer our points are, therefore we have a more well-formed cluster.

Code Implementation:

```
# Import required libraries and read data into a dataframe
import pandas as pd
df = pd.read_csv('wine-clustering.csv')
df.head()

# Do some data exploration
df.describe().T
df.info()
```

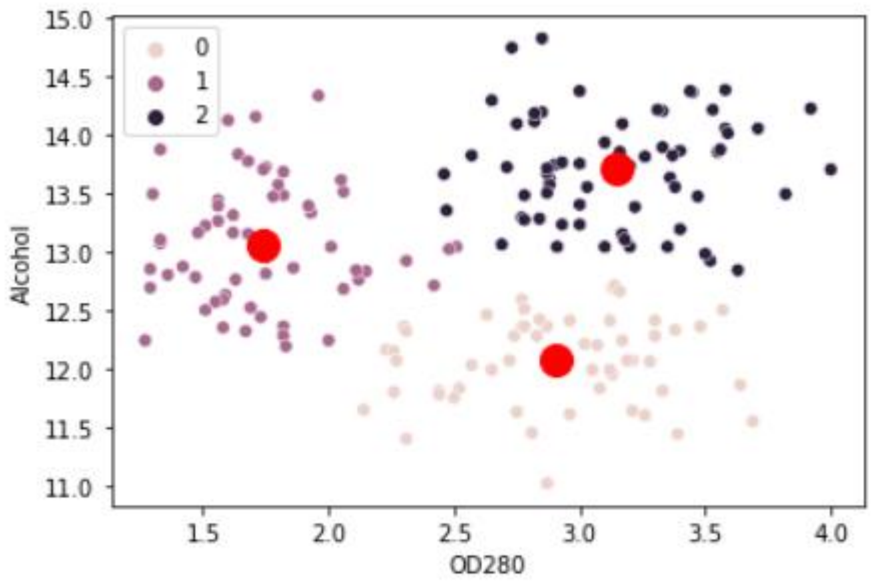
```
# Visual data exploration to identify correlation among columns of data
import seaborn as sns
sns.pairplot(df)

#Import algorithms from sklearn
from sklearn.cluster import KMeans

#the columns we will use for clustering are only two - the OD280 and Alcohol content of wines
selected_features = df[['OD280', 'Alcohol']]

# The random_state needs to be the same number to get reproducible results
kmeans_obj = KMeans(n_clusters=3, random_state=42)

# Fit the Kmeans algorithm on selected columns
kmeans_obj.fit(selected_features)
# Predict the cluster labels for data
y_kmeans = kmeans_obj.fit_predict(selected_features)

#Print the predicted labels
print(y_kmeans)

# Printing the cluster centers
centers = kmeans_obj.cluster_centers_
print(centers)

import matplotlib.pyplot as plt
#Visualize the Groups created
sns.scatterplot(x = selected_features['OD280'],
            y = selected_features['Alcohol'],
            hue=kmeans_obj.labels_)
#Visualize the cluster centroids
plt.scatter(kmeans_obj.cluster_centers_[:, 0],
        kmeans_obj.cluster_centers_[:, 1],
            s=200, c='red')
```

Sample Output:

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 2 0 0 0 2 0 1 0 1 2 2 2
 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

CENTERS:

```
[[ 2.90290909 12.07981818]
 [ 1.73741379 13.07413793]
 [ 3.14538462 13.71415385]]
```



Outcome of the Exercise:

The exercise provides a clear visualization of how the data is grouped into clusters, with each cluster's centroid representing its central point. The clustering helps reveal the inherent structure within the dataset, showing which data points are grouped together. Evaluation metrics such as inertia and silhouette scores can be used to assess clustering quality. The visual and numerical results guide further analysis or decision-making based on the clustering outcomes.

Viva/Interview Questions:

What are the main steps in the K-Means clustering algorithm?

Initialization, assignment, updating centroids, and repeating until convergence.

How do you determine the optimal number of clusters kkk **in K-Means?**

Using methods like the Elbow Method, Silhouette Score, or Gap Statistic.

What are the limitations of K-Means clustering?

Sensitivity to initialization, requires specifying kkk in advance, and assumes spherical clusters.

How does data standardization impact K-Means clustering?

Standardization ensures all features contribute equally to the distance metric, improving clustering performance.

What alternative algorithms can be used if K-Means is not suitable?

Alternatives include DBSCAN, hierarchical clustering, and Gaussian Mixture Models.

Program 9: Write a program to reduce the dimension of the dataset using principal component analysis (PCA).

Problem Statement: Reducing the dimension of the dataset.

Solution Overview: The solution involves reading the dataset. The PCA model is trained over the dataset to extract principal components. Finally, its reduced dimension is displayed with a suitable message.

Intuition: Principal Component Analysis (PCA) is a dimensionality reduction technique often used to reduce the size of a dataset while retaining as much of the original information as possible. The intuition behind PCA involves finding a new set of axes (called principal components) that capture the maximum variance in the data.

Steps in the Solution:

Data Collection:

Gather a dataset containing historical data along with any relevant features w.r.t to domain.

Data Preprocessing: (if any)

Handle any missing values in the dataset, as these can affect the accuracy of the model.

Encode categorical variables (e.g., location) using techniques such as one-hot encoding.

Normalize or standardize numerical features to ensure that all features contribute equally to the model.

Data standardization:

PCA is sensitive to the scale of the data for the features having different units or scales, it's important to standardize the data so that each feature has a mean of 0 and a standard deviation of 1.

Compute the Covariance Matrix:

The covariance matrix is calculated to understand the relationships between different features. It's an n x n where n is the number of features, and each element represents the covariance between two feature.

Compute the Eigenvectors and Eigenvalues:

The next step is to compute the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the directions (principal components) in which the data varies the most, and the eigenvalues represent the magnitude of the variance in these directions.

Sort Eigenvalues and Select Principal Components

Sort the eigenvalues in descending order and select the top k eigenvectors corresponding to the largest eigenvalues.

Project Data onto Principal Components:

Transform the original dataset by projecting it onto the new basis vectors (principal components). This involves multiplying the original standardized dataset by the matrix of the selected eigenvectors.

Code Implementation:

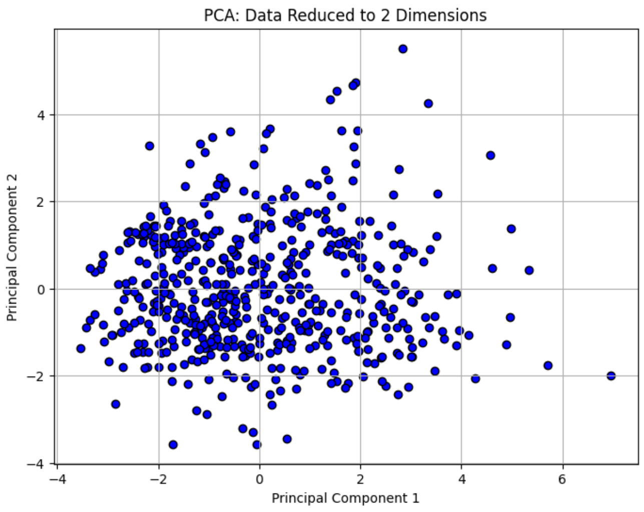
Dataset url: https://data.world/swarnapuri-sude/house-data

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

data = pd.read_csv('newhousing.csv')
print(data.shape)



# Standardize the data (important for PCA)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
reduced_data = pca.fit_transform(scaled_data)

# Create a DataFrame for the reduced data
reduced_df = pd.DataFrame(reduced_data, columns=['PC1', 'PC2'])

# Plot the reduced data
plt.figure(figsize=(8, 6))
plt.scatter(reduced_df['PC1'], reduced_df['PC2'], c='blue', marker='o', edgecolor='k')
plt.title('PCA: Data Reduced to 2 Dimensions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

# Display the reduced dimensions
print("\nReduced dimension:")
print(reduced_df)
print("The reduced dimension is ",reduced_df.shape)
```

Sample Output:

```
The reduced dimension is  (545, 2)
```

Sample Visualization:



Result = 0.7320342760357638

Outcome of the Exercise:

The outcome of the exercise is a trained PCA model for housing dataset. The results demonstrated the dimensionality reduction of the dataset used for classification.

Viva/Interview Questions:

Question: What is PCA??

Answer: PCA is a dimensionality reduction technique that transforms a dataset into a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Question: Why do we use PCA?

Answer: PCA is used to reduce the dimensionality of large datasets, improve computational efficiency, remove noise, and highlight important relationships in the data while retaining as much variability as possible.

Question: What are principal components?

Answer: Principal components are the new axes (or directions) formed after PCA transformation. They are linear combinations of the original features and are orthogonal to each other.

Question: Explain the steps involved in performing PCA.

Answer: MSE measures the average squared difference between the actual and predicted values, while RMSE is the square root of MSE, providing an error metric in the same units as the target variable.

Question: What is the significance of eigenvectors and eigenvalues in PCA?

Answer: Eigenvectors determine the directions of the new feature space (principal components), while eigenvalues determine the magnitude of variance in these directions. The eigenvector with the largest eigenvalue corresponds to the first principal component.

Program 10: Write a program to find the most specific hypothesis using Find-S algorithm. Read the training data from a .CSV file.

Problem Statement: finding the most specific hypothesis based on a given set of training data samples.

Solution Overview: Find S is a simple machine learning algorithm used for concept learning. Its primary goal is to find the most specific hypothesis that accurately classifies all positive examples in a given dataset**.**

Intuition:

Here are some key intuitions:

Incremental Learning: The algorithm learns from one positive example at a time. This incremental approach allows it to adapt to new information as it becomes available.

Greedy Generalization: When a positive example is encountered that contradicts the current hypothesis, the algorithm generalizes the hypothesis in the most minimal way possible. This greedy approach ensures that the hypothesis remains as specific as necessary to classify positive examples correctly.

Bias Towards Positive Examples: Find S is primarily concerned with correctly classifying positive examples. It does not explicitly consider negative examples, assuming that any hypothesis that correctly classifies all positive examples will also correctly classify negative ones.

Simplicity Preference: The algorithm tends to prefer simpler hypotheses over more complex ones. This bias towards simplicity can help to prevent overfitting, which occurs when a model becomes too closely tailored to the training data and performs poorly on unseen examples.

Steps in the Solution:

```
FIND-S Algorithm
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
        For each attribute constraint ai in h
                If the constraint ai is satisfied by x
                        Then do nothing
                Else replace ai in h by the next more general constraint that is satisfied
                by x
```

3. Output hypothesis h

Code Implementation:

```
import csv


num_attributes = 6
a = []
print("\n The Given Training Data Set \n")

with open('enjoysport.csv', 'r') as csvfile: reader = csv.reader(csvfile)
for row in reader:
        a.append (row)
        print(row)

print("\n The initial value of hypothesis: ") hypothesis = ['0'] * num_attributes
print(hypothesis)



for j in range(0,num_attributes): hypothesis[j] = a[0][j];

print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range(0,len(a)):
        if a[i][num_attributes]=='yes':
        for j in range(0,num_attributes):
                if a[i][j]!=hypothesis[j]:
                        hypothesis[j]='?'
                else :
                        hypothesis[j]= a[i][j]
print(" For Training instance No:{0} the hypothesis is ".format(i),hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

Sample Output:

```
The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
Find S: Finding a Maximally Specific Hypothesis For Training Example No:0 the hypothesis is
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No:1 the hypothesis is ['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:2 the hypothesis is 'sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No:3 the hypothesis is 'sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples:

['sunny', 'warm', '?', 'strong', '?', '?']
```

Outcome of the Exercise:

The final hypothesis suggests that the most specific conditions under which a positive outcome (e.g., a decision to play tennis) occurs are:

- The weather is Sunny.
- The temperature is Warm.
- The humidity and the wind condition don't matter (? represents any value).
- The wind is Strong.
- The rest of the features can be anything (? for Warm and Same).

Viva/Interview Questions:

Why is the Find-S algorithm called "Find-S"?

Answer: The algorithm is called "Find-S" because it finds the "most Specific" hypothesis that covers all positive examples in the training set. The "S" stands for "Specific."

What is the Find-S algorithm, and how does it work?

Answer: The Find-S algorithm is a simple machine learning algorithm used to find the most specific hypothesis that fits all the positive examples in a dataset. It starts with the most specific hypothesis and progressively generalizes it to include all positive instances. The algorithm only considers positive examples and ignores negative ones.

What are the limitations of the Find-S algorithm?

Answer: The main limitations of the Find-S algorithm include:

It only considers positive examples, which can lead to a hypothesis that is too specific and not generalizable.

7. Learning Resources:

Reference Books:

1. Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill.
2. Tom Mitchell, "Machine Learning", McGraw Hill Education.
3. Anand Rajaraman, Jure Leskovec and J.D. Ullman, “Mining of Massive Data sets”, ebook,Publisher, 2014 - www.mmds.org.
4. John D Kelleher, Brian Mac Namee, Aoife D'Arcy, "Fundamentals of Machine Learning for Predictive Data Analytics, Second Edition.

Web Based Resources and E-books:

1. NPTEL Course on “Introduction to Machine Learning ”, Prof. Balaraman R,

https://onlinecourses.nptel.ac.in/noc23_cs18/preview/

2. https://www.simplilearn.com/tutorials/machine-learning-tutorial/
3. https://github.com/mdozmorov/MachineLearning_notes/
4. Ebooks: https://machinelearningmastery.com/products/