24CY731 DATA MINING AND MACHINE LERANING IN CYBER SECURITY - LAB

1. a) Linear Regression

"A type of supervised ML algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data."

1.Import required libraries.

```
import pandas as pd
import numpy as np
```

2.Load data to a data frame.

```
data=pd.read_csv("Data.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Var 1   9 non-null      float64
 1   Var 2   9 non-null      float64
 2   Class   9 non-null      int64
dtypes: float64(2), int64(1)
memory usage: 348.0 bytes
```

3.Separate features and target.

```
x=np.array(data.drop("Class",axis=1))
y=data["Class"]
```

4. Split the data set in to training data and test data.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

5.Train the model.

```
model = LinearRegression()
model.fit(X_train, y_train)
```

6. Input the given features and predict the output.

```
new_input = np.array([[0.906, 0.606]])
predicted_value = model.predict(new_input)
print(f"Predicted value:{predicted_value}")
```

```
Predicted value:[0.86998495]
```

7.Convert the prediction to 0 or 1.

```
predicted_class = int(np.round(predicted_value))
print(f"Predicted class for var1=0.906, var2=0.606: {predicted_class}")
```

```
Predicted class for var1=0.906, var2=0.606: 1
```

1 b) Logistic Regression:

"Logistic regression is used for binary classification that takes input as independent variables and produces a probability value between 0 and 1."

1.Import necessary libraries.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

2.Read data from a dataset, load data to a data frame "data' and check for null values.

```
data=pd.read_csv("iris_data.csv")
data.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Here, we don't have any null values.

3.Identify features and targets.

```
x = data.drop("species", axis=1)
y = data['species']
```

Here, we are classifying the data based on the features such as sepal length, sepal width, petal length, petal width. And our target is find the corresponding species name.

4.Scale features.

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

5. Splits the dataset into training and testing sets.

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.3, random_state=42)
```

6.Initialize regression model.

```
model = LogisticRegression(max_iter=200, C=0.1)
```

(Sets the maximum number of iterations for the optimization algorithm).

7. Train the model and predict the value.

```
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

8. Find the accuracy and evaluate the model.

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.89
```

2. a) Import necessary libraries and load data.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
data=pd.read_csv('iris_data.csv')
```

shape() :

A method to find the dimension of the dataset. It returns number of rows and columns, enclosed in a paranthesis.

```
[3]: data.shape

[3]: (150, 5)
```

Here we have 150 number of rows and 5 number of columns.

info():

- This method prints information about the DataFrame.
- The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

```
[5]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Isnull().sum():

- isnull() method returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False.
- Sum() counts the number of true values.

```
[9]: data.isnull().sum()

[9]: sepal_length    0
     sepal_width     0
     petal_length    0
     petal_width     0
     species         0
     dtype: int64
```

describe():

- This method returns description of the data in the DataFrame.
- If the DataFrame contains numerical data, the description contains these information for each column:

```
[13]: data.describe()
```

[13]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

- count - Number of non empty values.
- mean – Average value.
- Std – Standard deviation.
- min - the minimum value.
- 25% - The 25% percentile.
- 50% - The 50% percentile.
- 75% - The 75% percentile.
- max - the maximum value.

Groupby():

- Used to split a DataFrame into groups based on one or more columns.

- Data is divided into groups, a function is applied to each group, and the results are combined into a new DataFrame.

```
[8]: data.groupby('species').mean()
```

[8]:

species	sepal_length	sepal_width	petal_length	petal_width
setosa	5.006	3.418	1.464	0.244
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

Here, we grouped the data based on the species. And the code returns mean value of each column for each category.

Splitting dataset into two sets:

- Feature (x) – Sepal length, sepal width, petal length, petal width.
- Target (y) - Species.
- For both training data and test data need x and y values.

```
x=data.drop("species",axis=1)
y=data['species']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Feature selection:

- Here, I am using correlation matrix as a feature selection method.
- Encode string datatype to numbers
- Set threshold=0.8
- Calculate correlation matrix.

```
data['species'] = LabelEncoder().fit_transform(data['species'])
correlation_matrix = data.corr()
high_correlation = correlation_matrix[correlation_matrix > 0.8]
print(high_correlation)
```

```
              sepal_length  sepal_width  petal_length  petal_width   species
sepal_length      1.000000          NaN      0.871754     0.817954       NaN
sepal_width            NaN          1.0           NaN          NaN       NaN
petal_length      0.871754          NaN      1.000000     0.962757  0.949043
petal_width       0.817954          NaN      0.962757     1.000000  0.956464
species                NaN          NaN      0.949043     0.956464  1.000000
```

- Filter the features having high correlation.

```
high_correlation = correlation_matrix[(correlation_matrix > 0.8) & (correlation_matrix != 1)]
print(high_correlation)
```

```
              sepal_length  sepal_width  petal_length  petal_width   species
sepal_length           NaN          NaN      0.871754     0.817954       NaN
sepal_width            NaN          NaN           NaN          NaN       NaN
petal_length      0.871754          NaN           NaN     0.962757  0.949043
petal_width       0.817954          NaN      0.962757          NaN  0.956464
species                NaN          NaN      0.949043     0.956464       NaN
```

2 c) **Decision tree classifier**:

```
dt_model = DecisionTreeClassifier(max_depth=2,random_state=42)
dt_model.fit(x_train, y_train)
```

```
DecisionTreeClassifier(max_depth=2, random_state=42)
```

```
y_pred=dt_model.predict(x_test)
```

```
accuracy=accuracy_score(y_test,y_pred)
print(f"Accuracy:{accuracy*100:.2f}%")
```

```
Accuracy:96.67%
```