Naan mudhalvan phase 5

Supervised learning in the context of classification involves training a machine learning model to categorize data into predefined classes or categories. To understand supervised classification thoroughly, let's delve into the theory and content related to it:

1. Problem Statement:

Classification is a type of supervised learning task where the goal is to assign input data points to discrete categories or classes.

It is used for tasks such as spam detection, sentiment analysis, image recognition, medical diagnosis, and more.

2. Labeled Data:

In supervised classification, you need a labeled dataset, which consists of input features and their corresponding class labels.

For example, in a binary classification task like email spam detection, the features could be email content, sender information, etc., and the labels are "spam" or "not spam."

3. Common Classification Algorithms:

Various classification algorithms can be used, depending on the problem and the data. Some common algorithms include:

Decision Trees

Random Forest

Support Vector Machines (SVM)

k-Nearest Neighbors (k-NN)

Naive Bayes

Logistic Regression

Neural Networks (Deep Learning)

4. Model Training:

During the training phase, the chosen classification algorithm learns to create a decision boundary that separates different classes based on the input features.

This is done by adjusting the model's internal parameters to minimize the classification error on the training data.

5. Evaluation Metrics:

To assess the performance of a classification model, several evaluation metrics are used, including:

Accuracy: The ratio of correctly classified instances to the total number of instances.

Precision: The proportion of true positive predictions among all positive predictions.

Recall: The proportion of true positive predictions among all actual positive instances.

F1-Score: The harmonic mean of precision and recall, providing a balanced measure.

Confusion Matrix: A table that shows the true positives, true negatives, false positives, and false negatives.

6. Overfitting and Underfitting:

Models can suffer from overfitting (model is too complex and fits noise in the data) or underfitting (model is too simple and cannot capture the underlying patterns).

Techniques like cross-validation and regularization can help mitigate overfitting.

7. Hyperparameter Tuning:

Many classification algorithms have hyperparameters that can be fine-tuned to optimize model performance.

Hyperparameter tuning involves finding the best combination of hyperparameters to achieve the best model performance.

8. Feature Engineering:

Feature selection and engineering play a crucial role in classification tasks. It involves choosing the most relevant features and creating new ones if needed.

Dimensionality reduction techniques like Principal Component Analysis (PCA) can be used to reduce the number of features.

9. Imbalanced Datasets:

Handling imbalanced datasets, where one class has significantly more samples than the others, is an important consideration in classification tasks.

Techniques like oversampling, undersampling, and using different evaluation metrics can help address this issue.

10. Model Deployment:

Once the model is trained and evaluated satisfactorily, it can be deployed in real-world applications, such as software systems, websites, or mobile apps.

11. Interpretability:

Understanding and interpreting the model's decisions are crucial, especially in applications where human judgment is involved.

Some models, like decision trees, offer interpretable results.

12. Continuous Improvement:

Continuous monitoring and maintenance of the classification model are essential. Over time, data distributions may change, and the model may need retraining or updating.

Code:

```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset as an example
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target labels

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

# Create a classification model (Logistic Regression in this case)
```

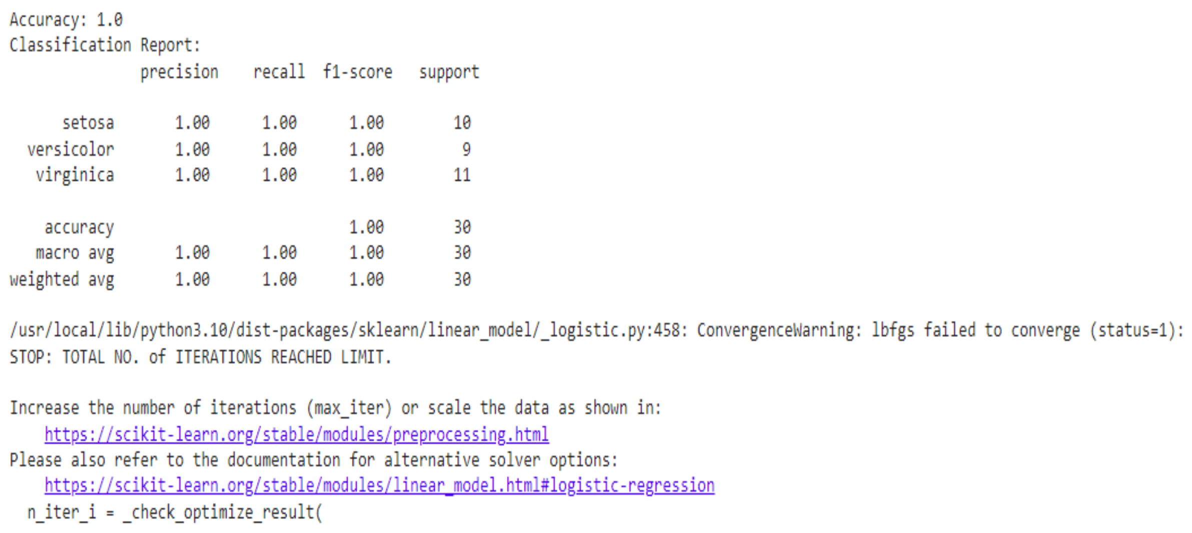
```python
model = LogisticRegression()

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

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred,
target_names=iris.target_names)

# Print the results
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", classification_rep)
```



```
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

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

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

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Accuracy – 1.0

Build & Deploy ML Application

How do you build and deploy a machine learning model?

Machine Learning Model Deployment Tutorial

Data Preprocessing. ...

Model Optimization and Training. ...

Model Serialization. ...

Prepare the Deployment Environment. ...

Build The Deployment API. ...

Test And Validate The Deployment. ...

Deploy The ML Model. ...

Monitor And Maintain The Deployment.

What is Model Deployment in Machine Learning?

Model deployment in machine learning means integrating a trained machine-learning model into a real-world system or application to automatically generate predictions or perform specific tasks. For example, imagine a healthcare company developing a model to predict the chances of readmission for patients with chronic diseases. Model deployment would involve taking the trained model and implementing it within the company's existing electronic health record system. Once deployed, the model can analyze patient data in real-time, offering insights to healthcare professionals to help them identify high-risk patients and take proactive measures to avoid patient readmissions.

Let us now move on and further explore how to deploy ML models in production using various frameworks.

ML Analysis: Application Layer DoS Attack Dataset

Dataset link: https://www.kaggle.com/code/hamzasamiullah/ml-analysis-application-layer-dos-attack-dataset/notebook

```
from mpl_toolkits.mplot3d import Axes3D #For Basic ploting
from sklearn.preprocessing import StandardScaler #Preprocessing
from sklearn import preprocessing    # Preprocessing
from sklearn.naive_bayes import GaussianNB #import gaussian naive bayes
model
from sklearn.tree import DecisionTreeClassifier #import Decision tree
classifier
from sklearn import metrics  #Import scikit-learn metrics module for
accuracy calculation
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/train_mosaic.csv
/kaggle/input/test_mosaic.csv
```

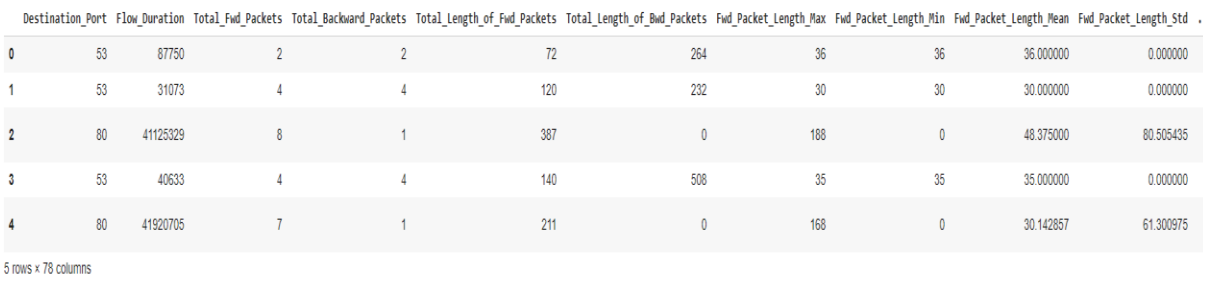
```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] <
50]] # For displaying purposes, pick columns that have between 1 and 50
unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow),
dpi = 80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep 
columns where there are more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or 
constant columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, 
facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()

nRowsRead = 1000 # specify No. of row. 'None' for whole data
# test_mosaic.csv may have more rows in reality, but we are only 
loading/previewing the first 1000 rows
df1 = pd.read_csv('/kaggle/input/test_mosaic.csv', delimiter=',', nrows 
= nRowsRead)
df1.dataframeName = 'test_mosaic.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 1000 rows and 78 columns
```

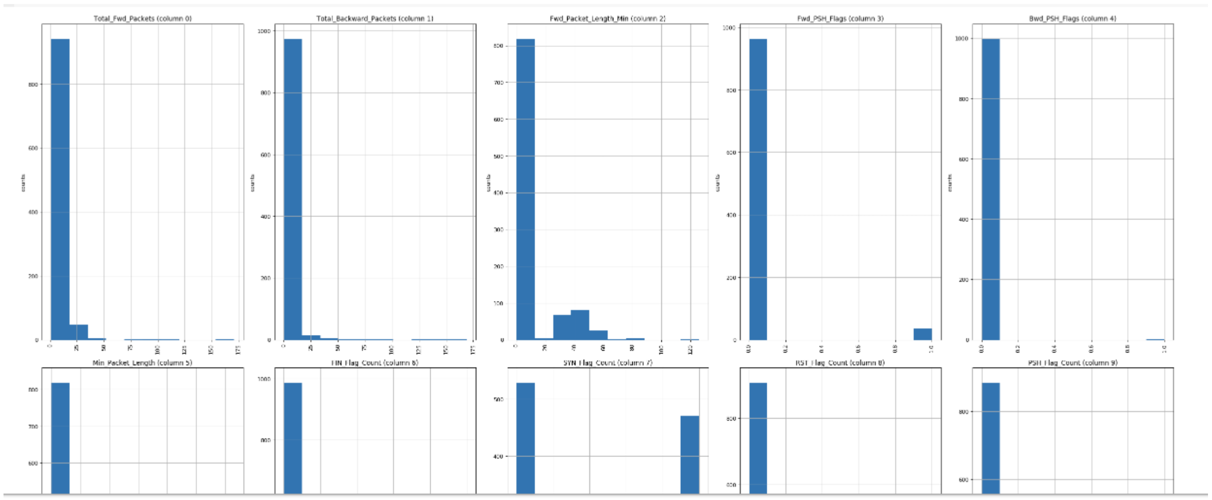
```
df1.head(5)
```



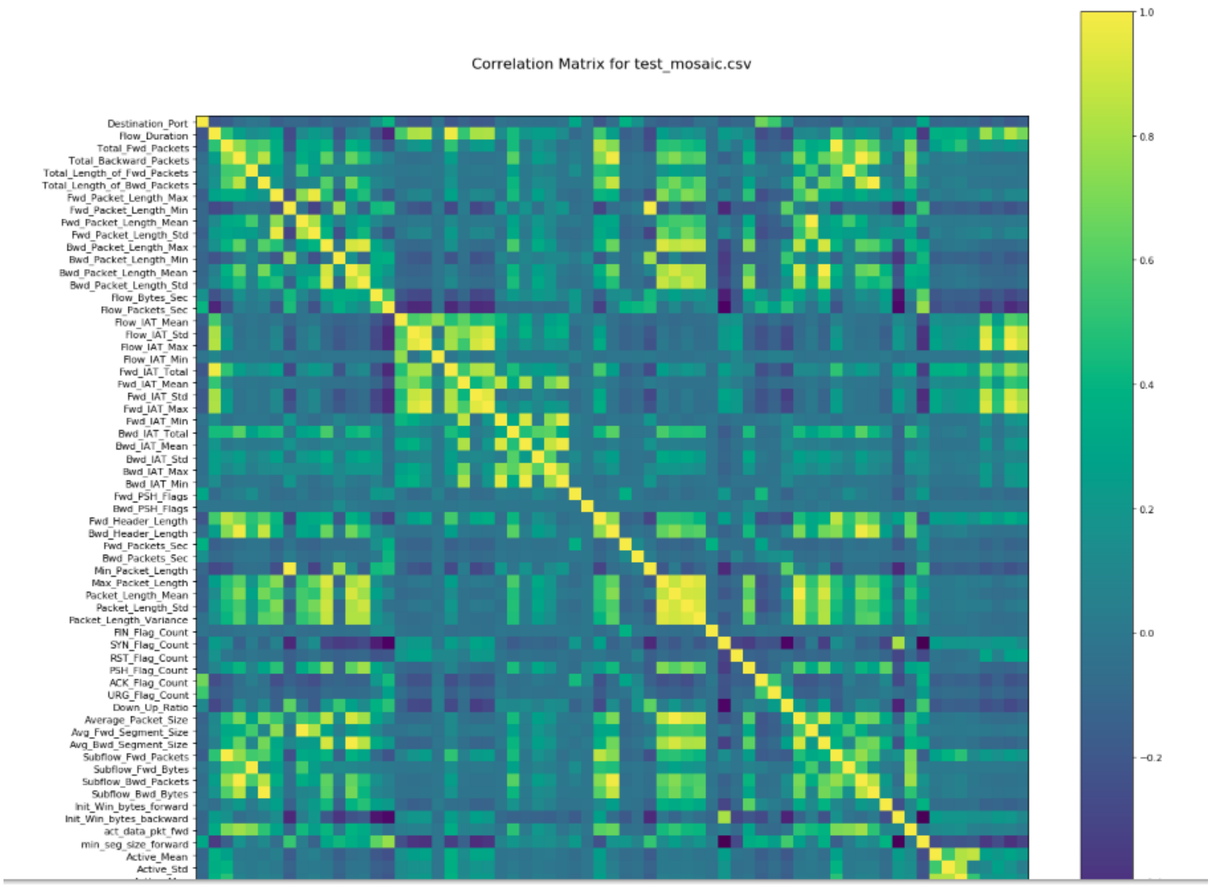
	Destination_Port	Flow_Duration	Total_Fwd_Packets	Total_Backward_Packets	Total_Length_of_Fwd_Packets	Total_Length_of_Bwd_Packets	Fwd_Packet_Length_Max	Fwd_Packet_Length_Min	Fwd_Packet_Length_Mean	Fwd_Packet_Length_Std	.
0	53	87750	2	2	72	264	36	36	36.000000	0.000000	
1	53	31073	4	4	120	232	30	30	30.000000	0.000000	
2	80	41125329	8	1	387	0	188	0	48.375000	80.505435	
3	53	40633	4	4	140	508	35	35	35.000000	0.000000	
4	80	41920705	7	1	211	0	168	0	30.142857	61.300975	

5 rows × 78 columns

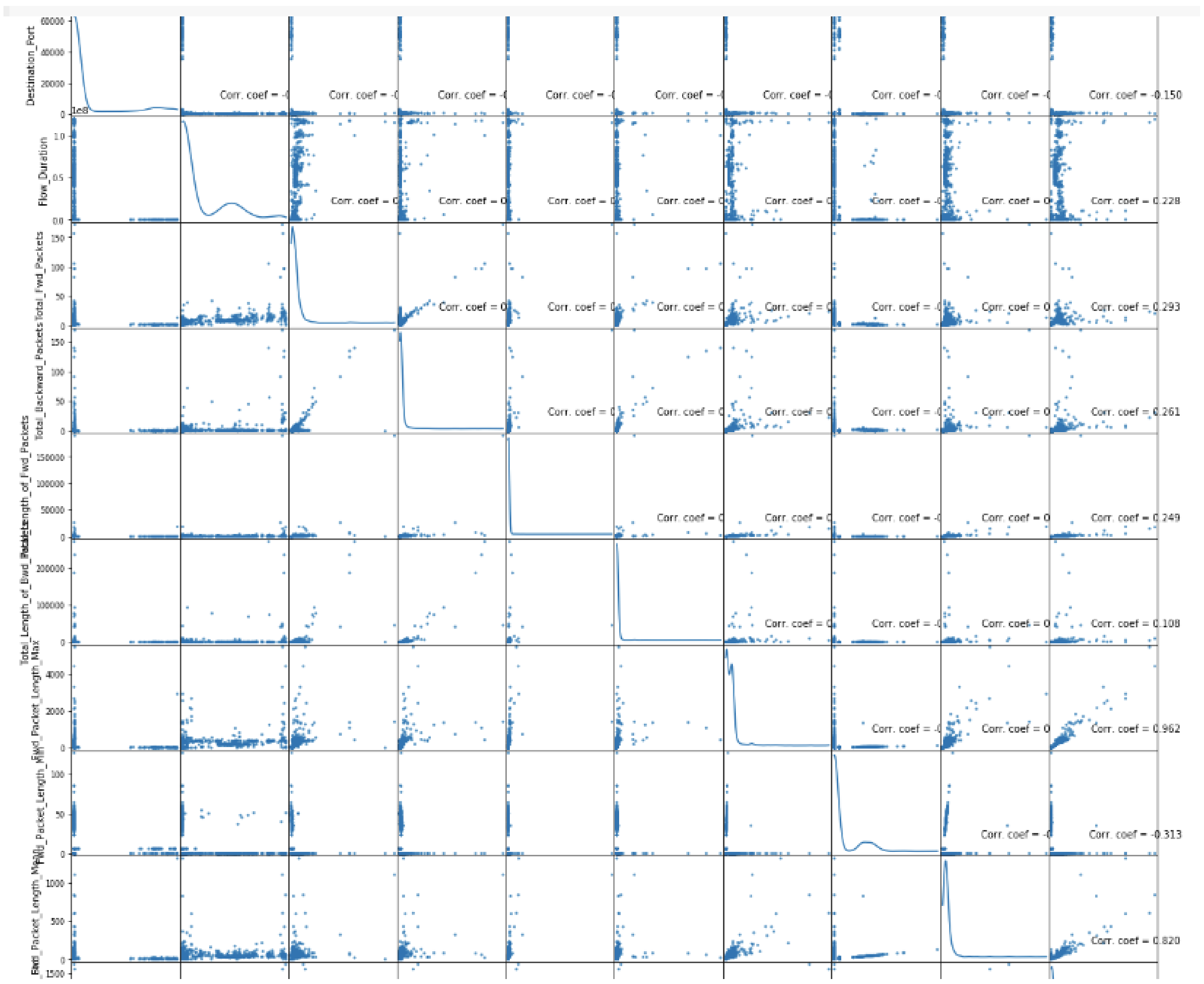
```
plotPerColumnDistribution(df1, 10, 5)
```



```
plotCorrelationMatrix(df1, 19)
```



```
plotScatterMatrix(df1, 20, 10)
```


Destination_Port
Flow_Duration
Total_Fwd_Packets
Total_Backward_Packets
Total_Length_of_Fwd_Packets
Total_Length_of_Bwd_Packets
Fwd_Packet_Length_Max
Fwd_Packet_Length_Min
Fwd_Packet_Length_Mean
Bwd_Packet_Length_Max
1e8
Corr. coef = -0.150
Corr. coef = 0.228
Corr. coef = 0.293
Corr. coef = 0.261
Corr. coef = 0.249
Corr. coef = 0.108
Corr. coef = 0.962
Corr. coef = -0.313
Corr. coef = 0.820