Lab 2: Classification with Logistic Regression, KNN, and MLP on NSL-KDD Dataset

By Djallel DILMI

Overview:

In this lab, students will apply three classification algorithms—Logistic Regression, K-Nearest Neighbors (KNN), and Multi-Layer Perceptron (MLP)—to the NSL-KDD dataset, a benchmark dataset for network intrusion detection. The aim is to classify network traffic as either normal or malicious.

Lab Objectives:

- Understand how to preprocess the NSL-KDD dataset for machine learning tasks.
- Apply Logistic Regression, K-Nearest Neighbors, and Multi-Layer Perceptron for classification.
- Evaluate and compare the performance of these models using classification metrics like accuracy, precision, recall, and F1 score.

Lab Tasks:

1. Download and Load the NSL-KDD Dataset:

- Download the **NSL-KDD** dataset from this link.
- Use the KDDTrain+.txt as the training set and KDDTest+.txt as the test set.
- Load the data into a pandas DataFrame.

```
import pandas as pd
# Load the NSL-KDD training and testing datasets
column_names = [...]  # Add the 43 column names from the NSL-KDD dataset
train_df = pd.read_csv('KDDTrain+.txt', header=None, names=column_names)
test_df = pd.read_csv('KDDTest+.txt', header=None, names=column_names)
```

TODO : write a dataset presentation that should include the dataset main objective.

2. Data Preprocessing:

1. **Label Encoding:**

- Convert the class labels (e.g., normal, anomaly) to binary values (0 for normal, 1 for anomaly) or perform multi-class classification if needed.

```
from sklearn.preprocessing import LabelEncoder
# Label encode the target
label_encoder = LabelEncoder()
train_df['label'] = label_encoder.fit_transform(train_df['label'])
test_df['label'] = label_encoder.transform(test_df['label'])
```

2. **Feature Scaling:**
    - Many machine learning algorithms, especially KNN and MLP, perform better when features are scaled to a uniform range.
    - Use StandardScaler to scale the numeric features.

```
from sklearn.preprocessing import StandardScaler
# Define the features and target
X_train = train_df.drop(columns=['label'])
y_train = train_df['label']
X_test = test_df.drop(columns=['label'])
y_test = test_df['label']
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. **Feature Selection (Optional):**
    - You may reduce the number of features using techniques such as PCA or simply by selecting a subset of the most important features.

3. Logistic Regression:

3.1. Train the Logistic Regression Model on the data
3.2. Make Predictions and Evaluate the Model:

4. K-Nearest Neighbors (KNN):

4.1. Train the KNN Model:
4.2. Make Predictions and Evaluate the Model:

5. Multi-Layer Perceptron (MLP):

5.1.Train the MLP Model:
Use MLPClassifier from scikit-learn for this task(to code by your self)
Hint search on the scikit learn website for MLPClassifier
Make Predictions and Evaluate the Model:

6. Model Comparison:

6.1. Compare Performance:
Compare the performance of all three models in terms of accuracy, precision, recall, and F1 score.
python
Copier le code
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_lr):.2f}")
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn):.2f}")
print(f"MLP Accuracy: {accuracy_score(y_test, y_pred_mlp):.2f}")
6.2. **Confusion Matrix:**

7. Code :

Translate your finding into a procedural code with python.