ELEC-E7130 Internet Traffic Measurements and Analysis

Assignment 8. Machine Learning

Name: Xingji Chen

Student ID: 101659554

E-mail: xingji.chen@aalto.fi

Aalto University
School of Electrical
Engineering

Task 1: Traffic classification

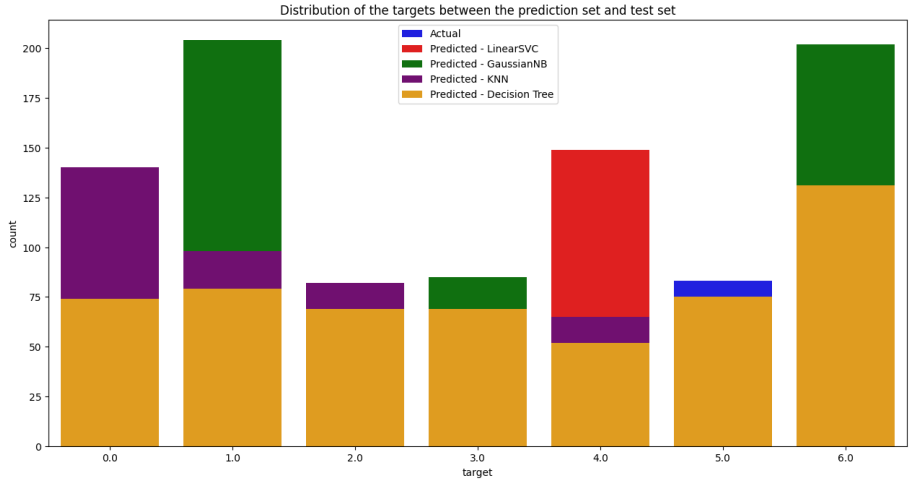
```
# IMPORT LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold


def main():
    # Step 1. Load CSV file
    df = pd.read_csv("datasets/combined_mix.csv")

    # Step 2. Split the data set into training set (80%) and test set
(20%)
    split_ratio = 0.8  # The split ratio should be a float
    split_index = int(split_ratio * len(df))
    train = df.iloc[:split_index]  # Training set
    test = df.iloc[split_index:]  # Test set

    # Step 3. Define the features and targets for both sets (training
and test)
    X_train = train.iloc[:, :-1]  # features (all except last column)
    Y_train = train.iloc[:, -1]  # targets (the last column)
    X_test = test.iloc[:, :-1]  # features (all except last column)
    Y_test = test.iloc[:, -1]  # targets (the last column)

    # Step 4. Train some ML models through different algorithms
    model1 = svm.LinearSVC(max_iter=100000, dual=False)  # 1.
LinearSVC model
    model1.fit(X_train, Y_train)
```

```
    model2 = GaussianNB()  # 2. Naive Bayes model
    model2.fit(X_train, Y_train)

    model3 = KNeighborsClassifier(n_neighbors=5)  # 3. KNN model
    model3.fit(X_train, Y_train)

    model4 = DecisionTreeClassifier()  # 4. Decision Tree Classifier
model
    model4.fit(X_train, Y_train)

    # Step 5. Perform classification (prediction) on an array of test
vectors X (features) for each model
    Y_predicted_model1 = model1.predict(X_test)
    Y_predicted_model2 = model2.predict(X_test)
    Y_predicted_model3 = model3.predict(X_test)
    Y_predicted_model4 = model4.predict(X_test)

    # Step 6. Plot the distribution of the targets between the
prediction set and test set (real data).
    plt.figure(figsize=(12, 8))
    sns.countplot(x=Y_test, color='blue', label='Actual')
    sns.countplot(x=Y_predicted_model1, color='red', label='Predicted
- LinearSVC')
    sns.countplot(x=Y_predicted_model2, color='green',
label='Predicted - GaussianNB')
    sns.countplot(x=Y_predicted_model3, color='purple',
label='Predicted - KNN')
    sns.countplot(x=Y_predicted_model4, color='orange',
label='Predicted - Decision Tree')
    plt.legend()
    plt.title('Distribution of the targets between the prediction set
and test set')
    plt.show()

    # Step 5. Evaluate the performance of each by calculating their
accuracy score
    print("-- Accuracy score: --")
    print("   1. LinearSVC model:              ",
accuracy_score(Y_test, Y_predicted_model1))
```

```
    print("   2. Naive Bayes model:            ",
accuracy_score(Y_test, Y_predicted_model2))
    print("   3. KNN model:                   ", accuracy_score(Y_test,
Y_predicted_model3))
    print("   4. Decision Tree Classifier model: ",
accuracy_score(Y_test, Y_predicted_model4))

    # Step 6. Setup K-fold CV and reset models. Using whole dataset
for K-fold CV.
    kf = KFold(n_splits=10)

    # Re-instantiate the models with the updated LinearSVC
    model1 = svm.LinearSVC(max_iter=100000, dual=False)
    model2 = GaussianNB()
    model3 = KNeighborsClassifier(n_neighbors=5)
    model4 = DecisionTreeClassifier()

    X = df.iloc[:, :-1]  # features (all except last column)
    Y = df.iloc[:, -1]  # targets (the last column)

    # Calculate the CV scores
    cv_scores_model1 = np.mean(cross_val_score(model1, X, Y, cv=kf))
    cv_scores_model2 = np.mean(cross_val_score(model2, X, Y, cv=kf))
    cv_scores_model3 = np.mean(cross_val_score(model3, X, Y, cv=kf))
    cv_scores_model4 = np.mean(cross_val_score(model4, X, Y, cv=kf))

    # Print the CV scores
    print("\n-- K-fold CV score: --")
    print("   1. LinearSVC model:             ", cv_scores_model1)
    print("   2. Naive Bayes model:           ", cv_scores_model2)
    print("   3. KNN model:                  ", cv_scores_model3)
    print("   4. Decision Tree Classifier model: ", cv_scores_model4)

    # Step 7. Save the most suitable model
    # Assuming the model with the highest cross-validation score as
the most suitable
    best_model = max([(model1, cv_scores_model1), (model2,
cv_scores_model2),
                      (model3, cv_scores_model3), (model4,
cv_scores_model4)],
```

Aalto University
School of Electrical
Engineering

```
                  key=lambda item: item[1])[0]

    with open('model_saved_ml_1.pkl', 'wb') as f:
        pickle.dump(best_model, f)


if __name__ == "__main__":
    main()
```




```
-- Accuracy score: --
   1. LinearSVC model:                  0.2877959927140255
   2. Naive Bayes model:                0.24408014571949
   3. KNN model:                        0.5737704918032787
   4. Decision Tree Classifier model:   0.7795992714025501

-- K-fold CV score: --
   1. LinearSVC model:                  0.26310948905109494
   2. Naive Bayes model:                0.21211280690112808
   3. KNN model:                        0.6060623755806237
   4. Decision Tree Classifier model:   0.790459190444592
```

- Compare the results, select the most suitable model for the dataset, and save it. Why did you choose this model?

  Upon comparing the results, we can draw the following conclusions. The Decision Tree Classifier model exhibits the highest accuracy on the test set (approximately 77.95%) and possesses the highest average cross-validation score (approximately 79.05%). The KNN model follows with around 57.38% accuracy on the test set and approximately 60.61% cross-validation score. In comparison, the LinearSVC and GaussianNB models perform poorly, with both test accuracy and cross-validation scores below 30%.

  Given these results, the Decision Tree Classifier model is the most suitable for this dataset. It not only has the highest accuracy but also maintains consistent performance across the folds of cross-validation, indicating the model's ability to generalize well to unseen data. Moreover, the high cross-validation score suggests that the model is not overfitting the training data and can be expected to perform well on similar unseen data. Therefore, the Decision Tree Classifier model should be selected and saved as the final model for further predictions on new data.

- What are the differences between accuracy score and k-Fold Cross-validation?

  The accuracy score is a measure of the proportion of correct predictions made by a model. It is calculated by dividing the number of correct predictions by the total number of predictions. Accuracy is commonly used in classification problems. It is easy to understand and provides a quick measure of model performance. However, it can be misleading, especially in datasets with imbalanced classes (where one class is much more frequent than others). It does not take into account the model's ability to handle new, unseen data.

  k-Fold Cross-Validation is a resampling procedure used to evaluate machine learning models on a limited data sample. This procedure has a parameter named k, which specifies the number of groups into which a given data sample is to be split. k-Fold Cross-Validation is often used in classification and regression problems. It provides a more robust measure of model performance. By dividing the dataset into k parts and using each part as a test set at some point, it ensures that every observation from the original dataset has the chance of appearing in both the training and test sets. This is particularly useful for small datasets and for estimating how well a model will generalize to an independent dataset. However, k-Fold Cross-Validation is more computationally expensive, especially as k increases. The interpretation of results from

k-Fold Cross-Validation can also be more complex than a simple accuracy score.

Aalto University
School of Electrical
Engineering

Task 2: Loading pre-defined model and evaluation

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score
from sklearn.metrics import classification_report

def main():
    # Step 1. Load CSV file as test set
    test = pd.read_csv("datasets/test_dataset.csv")

    # Step 2. Load ML models
    with open('LinearSVC_model_for_ml_2.pkl', 'rb') as f:
        model_loaded_1 = pickle.load(f)
    with open('GaussianNB_model_for_ml_2.pkl', 'rb') as f:
        model_loaded_2 = pickle.load(f)
    with open('KNeighborsClassifier_model_for_ml_2.pkl', 'rb') as f:
        model_loaded_3 = pickle.load(f)
    with open('DecisionTreeClassifier_for_ml_2.pkl', 'rb') as f:
        model_loaded_4 = pickle.load(f)

    # Step 3 Define the features and targets using whole set
    X_test = test.iloc[:, :-1]  # features (all except last column)
    Y_test = test.iloc[:, -1]  # targets (the last column)

    # Step 4. Perform classification (prediction) of the whole set
    Y_predicted_model1 = model_loaded_1.predict(X_test)
    Y_predicted_model2 = model_loaded_2.predict(X_test)
    Y_predicted_model3 = model_loaded_3.predict(X_test)
    Y_predicted_model4 = model_loaded_4.predict(X_test)

    # Step 5. Evaluate the models with confusion matrix
    print("-- Confusion matrix: --")
    print("   1. LinearSVC model:                \n",
```

```
confusion_matrix(Y_test, Y_predicted_model1))
    print("\n   2. Naive Bayes model:            \n",
confusion_matrix(Y_test, Y_predicted_model2))
    print("\n   3. KNN model:                   \n",
confusion_matrix(Y_test, Y_predicted_model3))
    print("\n   4. Decision Tree Classifier model: \n",
confusion_matrix(Y_test, Y_predicted_model4))

    # Step 6. Evaluate the models with accuracy score
    print("\n-- Accuracy score: --")
    print("   1. LinearSVC model:             ",
accuracy_score(Y_test, Y_predicted_model1))
    print("   2. Naive Bayes model:            ",
accuracy_score(Y_test, Y_predicted_model2))
    print("   3. KNN model:                  ", accuracy_score(Y_test,
Y_predicted_model3))
    print("   4. Decision Tree Classifier model: ",
accuracy_score(Y_test, Y_predicted_model4))

    # Step 7. Evaluate the models with precision and recall score
    # Note: If the target variable Y is multiclass, use
`average='macro'` or another averaging method suitable for multiclass
    print("\n-- Precision score: --")
    print("   1. LinearSVC model:             ",
precision_score(Y_test, Y_predicted_model1, average='macro'))
    print("   2. Naive Bayes model:            ",
precision_score(Y_test, Y_predicted_model2, average='macro'))
    print("   3. KNN model:                  ",
precision_score(Y_test, Y_predicted_model3, average='macro'))
    print("   4. Decision Tree Classifier model: ",
precision_score(Y_test, Y_predicted_model4, average='macro'))

    print("\n-- Recall score: --")
    print("   1. LinearSVC model:             ", recall_score(Y_test,
Y_predicted_model1, average='macro'))
    print("   2. Naive Bayes model:            ", recall_score(Y_test,
Y_predicted_model2, average='macro'))
    print("   3. KNN model:                  ", recall_score(Y_test,
Y_predicted_model3, average='macro'))
    print("   4. Decision Tree Classifier model: ",
```

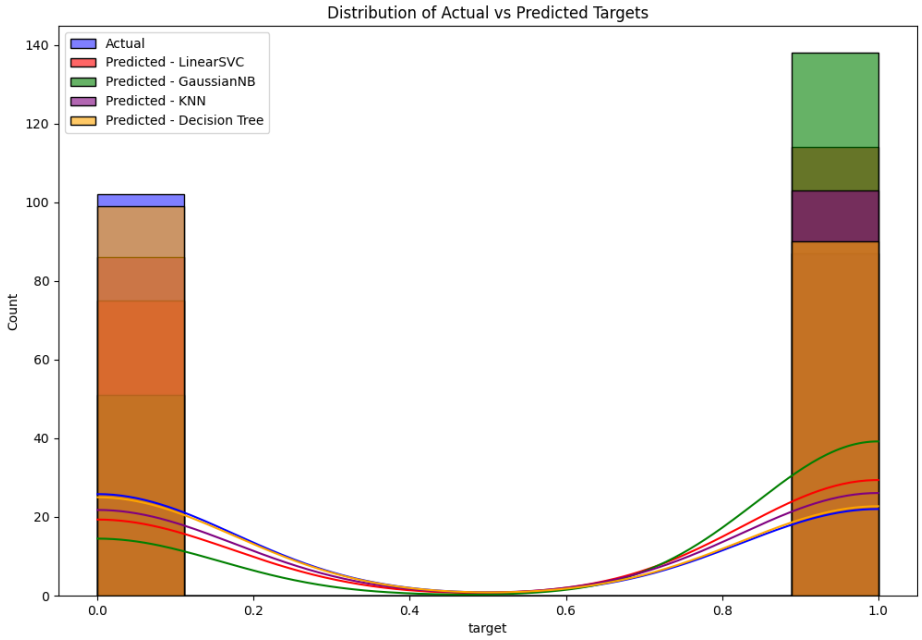
```
recall_score(Y_test, Y_predicted_model4, average='macro'))

    # Step 8. Evaluate the models with F1 score
    print("\n-- F1 score: --")
    print("   1. LinearSVC model:              ", f1_score(Y_test,
Y_predicted_model1, average='macro'))
    print("   2. Naive Bayes model:            ", f1_score(Y_test,
Y_predicted_model2, average='macro'))
    print("   3. KNN model:                   ", f1_score(Y_test,
Y_predicted_model3, average='macro'))
    print("   4. Decision Tree Classifier model: ", f1_score(Y_test,
Y_predicted_model4, average='macro'))

    # Step 9. Plot the distribution of the targets comparing between
the test set (real data) and prediction set (each model)
    plt.figure(figsize=(12, 8))
    sns.histplot(Y_test, color='blue', label='Actual', kde=True)
    sns.histplot(Y_predicted_model1, color='red', label='Predicted -
LinearSVC', kde=True, alpha=0.6)
    sns.histplot(Y_predicted_model2, color='green', label='Predicted
- GaussianNB', kde=True, alpha=0.6)
    sns.histplot(Y_predicted_model3, color='purple', label='Predicted
- KNN', kde=True, alpha=0.6)
    sns.histplot(Y_predicted_model4, color='orange', label='Predicted
- Decision Tree', kde=True, alpha=0.6)
    plt.legend()
    plt.title('Distribution of Actual vs Predicted Targets')
    plt.show()

if __name__ == "__main__":
    main()
```

This code is primarily used for the performance evaluation of machine learning models. It operates by loading a test dataset and pre-trained models, making predictions, and then assessing the performance of these models using various metrics.

```
-- Confusion matrix: --
   1. LinearSVC model:
 [[63 39]
 [12 75]]

   2. Naive Bayes model:
 [[44 58]
 [ 7 80]]

   3. KNN model:
 [[68 34]
 [18 69]]

   4. Decision Tree Classifier model:
 [[87 15]
 [12 75]]
```

```
-- Accuracy score: --
   1. LinearSVC model:                0.7301587301587301
   2. Naive Bayes model:              0.656084656084656
   3. KNN model:                      0.7248677248677249
   4. Decision Tree Classifier model: 0.8571428571428571

-- Precision score: --
   1. LinearSVC model:                0.7489473684210526
   2. Naive Bayes model:              0.7212276214833759
   3. KNN model:                      0.7303002935199819
   4. Decision Tree Classifier model: 0.8560606060606061

-- Recall score: --
   1. LinearSVC model:                0.7398580121703854
   2. Naive Bayes model:              0.6754563894523327
   3. KNN model:                      0.7298850574712643
   4. Decision Tree Classifier model: 0.8575050709939147

-- F1 score: --
   1. LinearSVC model:                0.7290665317480395
   2. Naive Bayes model:              0.6431372549019608
   3. KNN model:                      0.7248600223964166
   4. Decision Tree Classifier model: 0.8565646344548443
```

Aalto University
School of Electrical
Engineering



- Explain what information is provided by each performance metric.

1) The Confusion Matrix is used to describe the performance of classification models. It typically includes four parts: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). The Confusion Matrix helps in intuitively understanding the accuracy and types of errors made by the model in classifying each category.
2) The Accuracy Score is the most straightforward performance metric, representing the proportion of correctly classified samples to the total number of samples. Accuracy = (TP + TN) / (TP + TN + FP + FN). While accuracy is an intuitive measure of performance, it may not be a good indicator in imbalanced datasets.
3) The Precision Score is the proportion of actual positive cases among all samples predicted as positive by the model. Precision = TP / (TP + FP). This metric focuses on the accuracy of the model when it predicts a positive outcome.
4) The Recall Score is the proportion of samples correctly identified as positive by the model out of all actual positive samples. Recall = TP / (TP + FN). This metric focuses on the model's ability to cover actual positive cases.

- Which model seems to be most suitable for the dataset? Explain your reasons.

  Taking all the metrics into account, the Decision Tree Classifier is the most suitable model for this dataset. It not only has the highest accuracy but also maintains the best balance between precision and recall. This indicates that the Decision Tree model generalizes well and performs favorably on unseen data. The F1 score for the Decision Tree Classifier is also the highest, showing it has a good balance between precision and recall. Therefore, the Decision Tree Classifier is the most suitable for the dataset.

- Is relying solely on accuracy score sufficient when evaluating model performance? Please explain the reasons and share your insights.

  1) Relying solely on the accuracy score when evaluating model performance is insufficient. In datasets where one class significantly outnumbers others, a model can achieve high accuracy by always predicting the more frequent class, but this can be misleading.
  2) Additionally, accuracy does not differentiate between types of errors. It treats false positives and false negatives equally, but the cost of these errors can vary greatly in many practical scenarios. For example, in medical diagnosis, a false negative (misdiagnosing a sick person as healthy) is usually far worse than a false positive (misdiagnosing a healthy person as sick).
  3) Moreover, accuracy does not provide insight into the distribution of prediction results. A model might perform very well for one class but poorly for another.
  4) Model performance should be evaluated based on the specific requirements of the task, which often requires examining multiple metrics. Precision, recall, F1 score, and other metrics offer a more comprehensive understanding of model performance.

Aalto University
School of Electrical
Engineering

Task 3: Feature selection for classification purposes

```
import pandas as pd
import numpy as np
import time
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.simplefilter("ignore")

def evaluate_model(model, X_train, X_test, Y_train, Y_test):
    start = time.time()

    model.fit(X_train, Y_train)
    Y_predicted = model.predict(X_test)
    accuracy = accuracy_score(Y_test, Y_predicted)

    end = time.time()
    running_time = end - start

    return accuracy, running_time

def main():
    np.random.seed(1)

    df = pd.read_csv("datasets/combined_mix_nofs.csv")
    X = df.iloc[:, :-1]  # features (all except last column)
    Y = df.iloc[:, -1]  # targets (the last column)

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.1)

    selector = SelectKBest(f_classif, k=5)
```

```
    X_train_selected = selector.fit_transform(X_train, Y_train)
    X_test_selected = selector.transform(X_test)

    knn = KNeighborsClassifier()
    accuracy, runtime = evaluate_model(knn, X_train_selected,
X_test_selected, Y_train, Y_test)
    print("KNN with 5 features - Accuracy:", accuracy, "| Running
time:", runtime)

    accuracy, runtime = evaluate_model(knn, X_train, X_test, Y_train,
Y_test)
    print("KNN with all features - Accuracy:", accuracy, "| Running
time:", runtime)

    decision_tree = DecisionTreeClassifier()
    accuracy, runtime = evaluate_model(decision_tree,
X_train_selected, X_test_selected, Y_train, Y_test)
    print("Decision Tree with 5 features - Accuracy:", accuracy, "|
Running time:", runtime)

    accuracy, runtime = evaluate_model(decision_tree, X_train,
X_test, Y_train, Y_test)
    print("Decision Tree with all features - Accuracy:", accuracy, "|
Running time:", runtime)

if __name__ == "__main__":
    main()
```

This code evaluates the performance of K-Nearest Neighbors (KNN) and Decision Tree classifiers on a dataset for classification tasks. It includes feature selection and compares the impact of using a subset of features versus all features on the accuracy and running time of the models.

Results:

```
KNN with 5 features - Accuracy: 0.5127272727272727 | Running time: 0.008049726486206055
KNN with all features - Accuracy: 0.6072727272727273 | Running time: 0.0647742748260498
Decision Tree with 5 features - Accuracy: 0.6218181818181818 | Running time: 0.005000114440917969
Decision Tree with all features - Accuracy: 0.8145454545454546 | Running time: 0.020182371139526367
```

KNN with 5 features - Accuracy: 0.5127272727272727 | Running time: 0.008049726486206055
KNN with all features - Accuracy: 0.6072727272727273 | Running time: 0.0647742748260498

Decision Tree with 5 features - Accuracy: 0.6218181818181818 | Running time: 0.005000114440917969
Decision Tree with all features - Accuracy: 0.8145454545454546 | Running time: 0.020182371139526367

- Compare the performance of the models with and without feature selection using both algorithms and draw your conclusions.

  The results show that feature selection significantly impacts the performance and running time of both the K-Nearest Neighbors (KNN) and Decision Tree classifiers.

  For the KNN classifier, using only the top 5 features resulted in a lower accuracy (approximately 51.27%), while using all features yielded a higher accuracy (approximately 60.73%). However, the running time was significantly reduced to 0.008 seconds with only the top 5 features, compared to a longer running time of 0.065 seconds with all features.

  For the Decision Tree classifier, the accuracy with the top 5 features was around 62.18%, whereas using all features significantly increased the accuracy to approximately 81.45%. The running time with 5 features increased slightly from 0.005 seconds to 0.020 seconds when using all features, but this increase was not as substantial as with the KNN classifier.

  Compared to the KNN classifier, the Decision Tree classifier benefits more from using all available features, which indicates that the additional features provide meaningful information for the decision-making process within the model. The KNN classifier is more sensitive to feature selection. The reduced feature set leads to decreased accuracy, suggesting that KNN may rely on the complete set of features to make better estimations. Additionally, feature selection results in faster running times for both classifiers, which is particularly beneficial when computational resources or time are limited.

  It is important to note that there is a trade-off between accuracy and running time. Using all features is preferable when the highest accuracy is sought. However, if the speed of operation is more critical, feature selection can be beneficial.

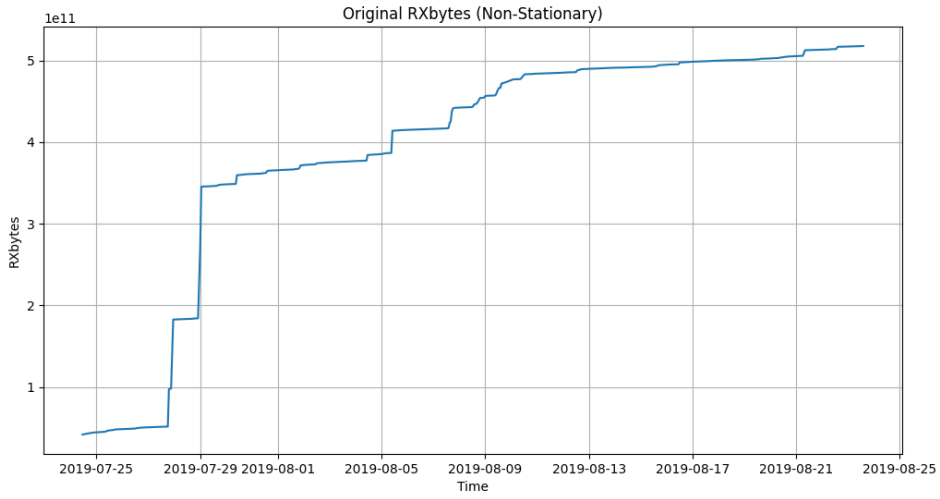
Aalto University
School of Electrical
Engineering

Task 4: Making non-stationary data into stationary

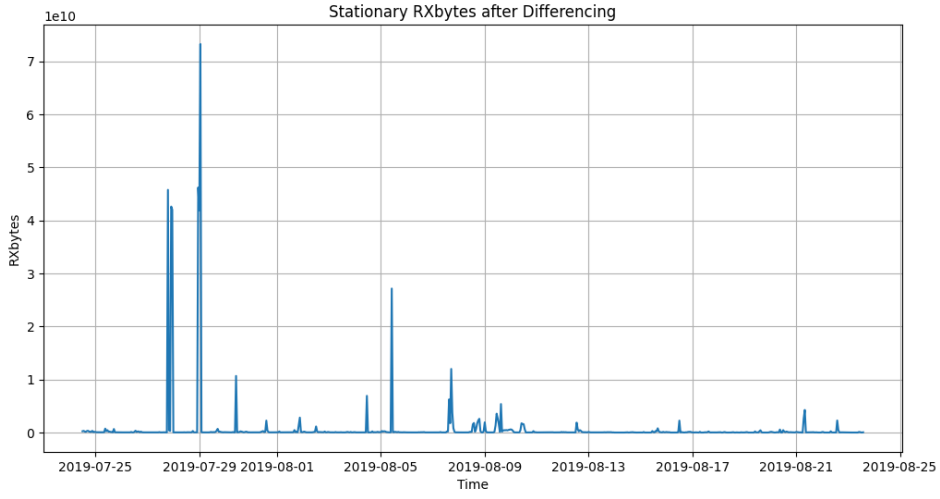
```
import pandas as pd
import matplotlib.pyplot as plt

def plot_time_series(data, title):
    plt.figure(figsize=(12, 6))
    plt.plot(data)
    plt.title(title)
    plt.xlabel('Time')
    plt.ylabel('RXbytes')
    plt.grid(True)
    plt.show()

def main():
    # Load the dataset
    df = pd.read_csv('datasets/bytes.csv', parse_dates=['Time'],
index_col='Time')

    # Select the 'RXbytes' column
    rx_bytes = df['RXbytes']

    # Plot original (non-stationary) data
    plot_time_series(rx_bytes, 'Original RXbytes (Non-Stationary)')

    # Make the data stationary using differencing
    stationary_rx_bytes = rx_bytes.diff().dropna()

    # Plot stationary data
    plot_time_series(stationary_rx_bytes, 'Stationary RXbytes after
Differencing')

if __name__ == "__main__":
    main()
```

The purpose of this code is to read time series data from a CSV file, display the graph of the original data, and process the data using the differencing method to make it stationary, i.e., to remove trends and seasonality from the time series data, and finally to display the graph of the processed data.

- Plot the received bytes (non-stationary data)



- Make the received bytes into stationary (drop possible NaNs that may occur) and plot them.



- Describe briefly their difference and the usefulness of stationary time series data.

The first image displays the original received bytes (non-stationary data). Non-stationary data often contain trends, seasonality, or other structures dependent on time. A trend is clearly visible in the non-stationary data, indicating an overall increase in received bytes over time.

This makes modeling the time series difficult because the fundamental characteristics of the data may change over time.

The second image shows the received bytes after applying a differencing method to stabilize the data. This process typically involves subtracting the previous observation from the current observation. By doing so, trends and seasonality are removed, which is reflected in the second image where the data no longer exhibits a long-term trend but instead fluctuates around a constant mean.

Comparing the two types of data, stationary data are often more useful because most time series modeling techniques assume or require the data to be stationary. This implies that the statistical properties of the series (mean, variance, autocorrelation, etc.) remain constant over time, simplifying the modeling process. Stationary time series data are easier to predict since the future values of the series have a relationship with past values. Moreover, stationary data allow models to learn patterns in the data without the added complexity of trends and seasonality.

Aalto University
School of Electrical
Engineering

Task 5: Making time series data into supervised problem

```python
import pandas as pd

def series_to_supervised(data, n_in=1, n_out=1):
    df = pd.DataFrame(data)
    cols, names = list(), list()

    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('t-%d' % i)]

    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('t')]
        else:
            names += [('t+%d' % i)]

    agg = pd.concat(cols, axis=1)
    agg.columns = names

    return agg

df = pd.read_csv('datasets/rtt.csv')
values = df['RTT'].values

df_supervised_1 = series_to_supervised(values, 1)
df_supervised_3 = series_to_supervised(values, 3)
df_supervised_5 = series_to_supervised(values, 5)

print("Window size of 1 :\n", df_supervised_1.head(6))
print("\nWindow size of 3 :\n", df_supervised_3.head(6))
print("\nWindow size of 5 :\n", df_supervised_5.head(6))
```

This code primarily serves to reformat time series data for use in supervised learning. Initially, the code reads a CSV file containing time series data and transforms a specified column into a new data frame to create a lagged dataset.

Within the `series_to_supervised` function, the code initially generates an input sequence, which are the n previous data points before the current observation, serving as input features for the model. Subsequently, it produces an output sequence, which are the n future data points starting from the current time point, intended as the target for the model to predict. In this manner, the model is capable of learning patterns to predict future data based on past data. To demonstrate how this transformation varies with different window sizes (i.e., different numbers of lag observations), the code creates three separate supervised learning datasets with window sizes of 1, 3, and 5 lags. Finally, the code prints the first six rows of each dataset.

```
Window size of 1:
      t-1    t
0     NaN  220
1   220.0  178
2   178.0  175
3   175.0  168
4   168.0  171
5   171.0  167

Window size of 3:
      t-3    t-2    t-1    t
0     NaN    NaN    NaN  220
1     NaN    NaN  220.0  178
2     NaN  220.0  178.0  175
3   220.0  178.0  175.0  168
4   178.0  175.0  168.0  171
5   175.0  168.0  171.0  167

Window size of 5:
      t-5    t-4    t-3    t-2    t-1    t
0     NaN    NaN    NaN    NaN    NaN  220
1     NaN    NaN    NaN    NaN  220.0  178
2     NaN    NaN    NaN  220.0  178.0  175
3     NaN    NaN  220.0  178.0  175.0  168
4     NaN  220.0  178.0  175.0  168.0  171
5   220.0  178.0  175.0  168.0  171.0  167
```

- Assuming the function is doing one-step-predictions, which column would be features and which column would be targets in all these cases?

  Assuming the function is set up for one-step predictions, for each of the cases, the features would be the columns representing the past values, and the target would be the column representing the current value.

  When the size of window is 1, features is t - 1, target is t.

  When the size of window is 3, features is t – 3, t – 2, t - 1, target is t.

  When the size of window is 5, features is t – 5, t – 4, t – 3, t – 2, t - 1, target is t.

- What is the importance to convert into supervised learning?

  Supervised learning models require input features and target variables for training. By converting time series data into this format, we can apply a broad range of supervised learning algorithms to make predictions based on past observations. This enables the model to identify patterns within the data sequence. By framing the issue as a supervised learning task, the model can learn the mapping from past observations (features) to future values (targets).

  Many machine learning algorithms are designed to solve supervised learning problems. Converting time series data into this form makes it compatible with these algorithms, which may not be inherently suited for raw time series data. It allows for the use of standard evaluation metrics and validation techniques to assess model performance, such as mean squared error and mean absolute error, providing a clear framework for model comparison and selection.

  By adjusting the number of lag observations (input) and the prediction horizon (output), different types of forecasting models can be created, such as single-step or multi-step forecasts. The supervised learning format aids models in generalizing from past data to make predictions on new, unseen data, which is the ultimate goal of most predictive modeling tasks.