

CS 345 Final Project – Fall 2025

Introduction

For this project, you will analyse network traffic data again. Specifically, you will use the dataset `CS345FinalProject.zip`, which contains seven individual `.csv` files, and may refer to the network traffic project we completed in class to help you implement the tasks described below (the solutions to the in-class network traffic analysis project are available on WebCampus). The detailed requirements for each part of the project are given in the corresponding sections. You should implement your code on your local machine, since you will work with several different Python files. Please make sure that all necessary packages (`numpy`, `pandas`, `scikit-learn`, etc.) are installed.

Due Dec. 12, 2025, 11:59pm

You will need to submit a `report.pdf` file that contains all evaluation results as specified throughout the project description, a `helpers.py` file, a `multiclass_classification.py` file, and a `main.py` file that implements the functions required in the following sections.

For any questions regarding the final project, please contact TA Jiahao (jiahaox@unr.edu) first instead of emailing the instructor.

1 Helper Functions (25 points)

You are required to implement the following helper functions in a file named `helpers.py`. These functions will be used throughout the entire project for data loading, preprocessing, dataset splitting, and model evaluation. Therefore, the function names, input arguments, and return types must strictly follow the specifications below.

`load_data(fname)` (5 points)

This function takes the path to a CSV file as input and loads the data into a `pandas DataFrame`. The loaded `DataFrame` must be returned.

`clean_data(df)` (5 points)

This function takes a `pandas DataFrame` and performs basic data cleaning. At a minimum, it must replace any `NaN`, `+Inf`, and `-Inf` values with valid numerical values (for example, replacing them with 0, the mean, or the median). You must clearly state your chosen strategy in your documentation. The function returns the cleaned `DataFrame`.

`split_data(df, ratio)` (5 points)

This function takes a **pandas DataFrame** and randomly splits it into training and testing sets according to the given ratio. The function must return two **DataFrames**: `df_train` and `df_test`. The splitting strategy should be clearly stated.

`model_evaluation(model, X_test, y_test)` (10 points)

This function evaluates a trained classification model on the testing set. It must, at a minimum, perform the following steps:

- Use the trained model to generate predictions on `X_test`.
- Compute and print the test accuracy.
- Compute and print the confusion matrix.
- Compute and print the classification report, including precision, recall, and F1-score for each class.

Hint: You may need to use the following functions from **scikit-learn**: `accuracy_score`, `confusion_matrix`, and `classification_report`.

The printed evaluation results will be used directly in the final project report.

2 Multi-Class Classification (55 points)

Here, you will perform multi-class classification using the network traffic data. You will build and compare two approaches: *direct multi-class classification* and *hierarchical multi-class classification*.

Implement the required functions for this part in a file named `multiclass_classification.py`.

2.1 Direct Multi-Class Classification (15 points)

In this part, you will choose a machine learning model introduced in class (for example, a multi-layer perceptron or a decision tree) to directly perform multi-class classification on the full set of classes.

Implement the following function:

`direct_multiclass_train(model_name, X_train, y_train)`

This function takes `model_name` (e.g., "mlp" or "dt") as input, together with the training data `X_train` (features) and `y_train` (labels). It should:

- construct the chosen model (for example, `MLPClassifier` when `model_name="mlp"`),
- fit the model on the training data, and

- return the trained model.

In your `main.py`, after training, you must call the previously defined `model_evaluation` function to evaluate the model on the testing set and record the results in `report.pdf`.

2.2 Hierarchical Multi-Class Classification (40 points)

In this part, you will build a hierarchical classifier for network traffic. Instead of predicting all classes at once, you will first decide whether a record is benign or malicious, then classify the type of malicious activity, and finally distinguish between the different web attack types. All functions for this part must be implemented in `multiclass_classification.py`.

During each layer, you should use the helper functions from the previous sections (for example, `direct_multiclass_train` and `model_evaluation`) to train and evaluate the models on the corresponding training and testing sets.

2.2.1 First layer: Binary classification (benign vs. malicious)

In the first layer, you will convert the original multi-class dataset into a binary classification problem with two labels: `BENIGN` and `MALICIOUS`. You must handle the strong class imbalance between benign and malicious samples by applying a sampling technique on the training set (for example, under-sampling or over-sampling).

`get_binary_dataset(df)` (5 points)

This function takes a `pandas DataFrame` `df` containing the original labels and returns a new `DataFrame` in which the label column has only two values: `BENIGN` and `MALICIOUS`.

In addition, you must implement a sampling function for the binary training set:

`data_resampling(df_binary_train)` (10 points)

This function takes a binary training `DataFrame` `df_binary_train` (with labels `BENIGN` and `MALICIOUS`) and returns a new `DataFrame` where the class imbalance has been reduced using a sampling technique. The function should:

- separate features and labels from `df_binary_train`,
- apply a sampling method to obtain a more balanced dataset, and
- return the resampled `DataFrame` with the same columns as the input.

In your `main.py`, you should:

- call `get_binary_dataset` on the training and testing sets,
- apply `data_resampling` to the binary training set to obtain a balanced binary dataset,
- train a binary classifier with `direct_multiclass_train`, and
- evaluate it using `model_evaluation`.

The evaluation results for this first layer must be included in [report.pdf](#).

2.2.2 Second layer: Multi-class classification on malicious samples

In the second layer, you will focus only on malicious samples. The goal is to classify the type of malicious activity. At this stage, all web attack types should be merged into a single class (for example, "Web Attack"), while other attack types keep their original labels.

`get_malicious_dataset_with_merged_webattacks(df)` (5 points)

This function takes a `pandas DataFrame df` and returns a new `DataFrame` that contains only malicious samples. All benign samples should be removed. Among the malicious samples, the three web attack types must be merged into a single web attack class label (for example, 'Web Attack'), while all other attack types remain as separate classes.

In your [main.py](#), you should call this function on the training and testing sets, train a multi-class model with `direct_multiclass_train`, and evaluate it with `model_evaluation`. The performance of this second-layer classifier must be reported in your [report.pdf](#).

2.2.3 Third layer: Web attack classification

In the third layer, you will zoom in on web attack samples only and distinguish between different web attack subtypes (for example, brute force, XSS, and SQL injection).

`get_webattack_dataset(df)` (5 points)

This function takes a `pandas DataFrame df` and returns a new `DataFrame` that contains only web attack samples. The label column should keep the original three web attack subtype labels, so that a multi-class model can be trained to distinguish between them.

In your [main.py](#), you should call this function on the training and testing sets, train a multi-class model on the web attack subtypes using `direct_multiclass_train`, and evaluate it with `model_evaluation`. The performance of this third-layer classifier must also be included in [report.pdf](#).

2.2.4 Final hierarchical evaluation

After training the three models for the first, second, and third layers, you will combine them into a single hierarchical prediction function that operates on the original test set.

```
AI_driven_network_traffic_analysis(model_1st, model_2nd,  
model_3rd, X_test, y_test) (15 points)
```

This function takes the three trained models (for the first, second, and third layers) and the original testing set `X_test`, `y_test`. It should:

- first use the first-layer model (`model_1st`) to predict whether each record is `BENIGN` or `MALICIOUS`;
- for records predicted as `MALICIOUS`, apply the second-layer model (`model_2nd`) to predict the attack type, where web attacks are treated as a single merged class;
- for records predicted as web attacks in the second layer, apply the third-layer model (`model_3rd`) to predict the specific web attack subtype.

The function should produce the final hierarchical predictions for all records in `X_test`, compute and print the test accuracy, confusion matrix, and classification report in [report.pdf](#).

3 Report and Code Quality (20 points)

The remaining 20 points of the final project will be awarded based on the overall quality, completeness, and readability of your report and code implementation. You must provide a well-organized `main.py` that correctly integrates all the functions you developed in previous sections and demonstrates a complete experimental pipeline.

All experimental results generated throughout the project (including those from direct multi-class classification, each layer of the hierarchical classification, and the final hierarchical system) must be clearly reported in the final PDF report. The report should be well-organized and easy to follow.

END