Mini Project #4: Secure System Analysis Using Machine Learning

CS 6263/ECE 8813: Cyber-Physical Systems Security (Summer 2025)

Assigned: July 2, 2025 **Due:** July 16, 2025 11:59pm EST

Objective

As we learned in this course, one of the prevalent attack methodologies in cyber-physical systems is false data injection, wherein an attacker alters the sensor readings to manipulate system behavior or obfuscate malicious undertakings. In electric power systems, for instance, false data about an overloaded branch (i.e., transmission line) can lead to dire repercussions. If operators believe a branch is overloaded, they might reroute power or take it offline as a precaution. However, this could cause an unintended power imbalance elsewhere. Other sections of the grid might then face genuine overload, initiating shutdowns. This can trigger a domino effect, where one shutdown leads to subsequent ones in adjoining sections. The resulting cascade can cause widespread blackouts, affecting critical infrastructure, hospitals, and homes, with serious economic and safety implications.

In this project, through the lens of machine learning, we aim to predict and validate the genuine behavior of a cyber-physical system, enabling the identification of discrepancies that might arise from cyber-physical attacks.

We have prepared two one-hour sessions to help you ramp up on the ML topics and their applications in the CPS security domain. Before moving forward, we strongly recommend watching these sessions.

Cyber-Physical Electric Power Systems

In this project, an electric power system will be examined as our cyber-physical system. Electric power systems are designed to deliver electricity from producers to consumers in an efficient and reliable manner. They are composed of three main parts: generation, transmission, and distribution (see Figure 1). Generation involves the production of electricity from various energy sources. Transmission section is responsible for transporting high-voltage electricity over long distances from power plants to substations near demand centers. Distribution then takes over, reducing the voltage through transformers and delivering the power to consumers through a network of local lines. A *bus* (or node) in electric power systems is a central junction point that connects multiple branches of the network at a common voltage level. In mathematical terms, an electric power network can be modeled as a graph $\mathcal{G} = (\mathcal{B}, \mathcal{L})$, where $\mathcal{B}$ represents the set of nodes, which are the buses[1], and $\mathcal{L}$ represents the set of edges, which are the transmission

[1]In this project, the terms “bus” and “node” are used interchangeably.

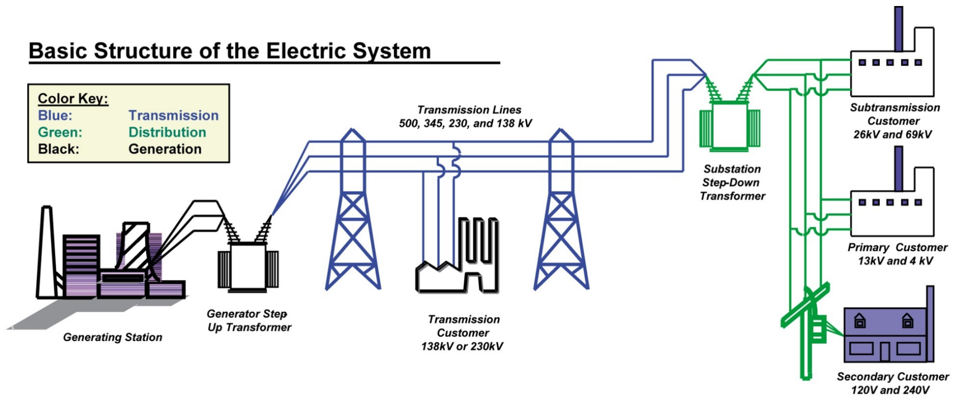
Basic Structure of the Electric System
Color Key:
Blue: Transmission
Green: Distribution
Black: Generation
Transmission Lines 500, 345, 230, and 138 kV
Substation Step-Down Transformer
Subtransmission Customer 26kV and 69kV
Primary Customer 13kV and 4 kV
Secondary Customer 120V and 240V
Generating Station
Generator Step Up Transformer
Transmission Customer 138kV or 230kV

Figure 1: Power system components: generation, transmission, and distribution

lines[2].

Electric power systems are one of the most critical infrastructures that are increasingly becoming targets for cyber-attacks. In 2015, a cyberattack on Ukraine's power grid left over 200,000 residents without electricity, marking one of the first power outages attributed to a cyber-physical attack[3]. Another incident in 2019 saw a Western U.S. utility experiencing a denial-of-service attack that momentarily disrupted grid operations[4].

System Under Study: Overview, Data, and Sensor Assumptions

In this project, we examine a standard 39-node system depicted in Figure 2, which includes 10 generators and 46 transmission lines (branches). We base our analysis on data collected by various sensors installed throughout the system. These sensors are crucial for gathering information, and we categorize the data they collect into two main types, each with its own security implications:

1. *Demand and Generation Data at Buses:* This category consists of data gathered by reliable sensors placed at different buses within the system. These sensors provide real-time and accurate information about power demand and generation at these nodes. The reliability of these sensors is ensured, making the data they provide a trustworthy foundation for system analysis. These sensors guarantee a high level of confidentiality, integrity, and availability.

2. *Transmission Line Overload Status:* We assume unlike the demand and generation data, the information regarding the status of transmission line overloads is more susceptible to security threats, including cyber-attacks. There is a risk that this data could be manipulated, leading to incorrect reporting of overload statuses. It is crucial to scrutinize and verify this data to protect the system from the repercussions of any such attacks. Here, the assumption is that the availability and integrity of these sensors are at risk.

[2]In this project, the terms "transmission line" and "branch" are used interchangeably.
[3]Lee, R. M., Assante, M. J., & Conway, T. (2016). Analysis of the cyber attack on the Ukrainian power grid. E-ISAC.
[4]U.S. Department of Energy. (2019). Electric Emergency Incident and Disturbance Report.

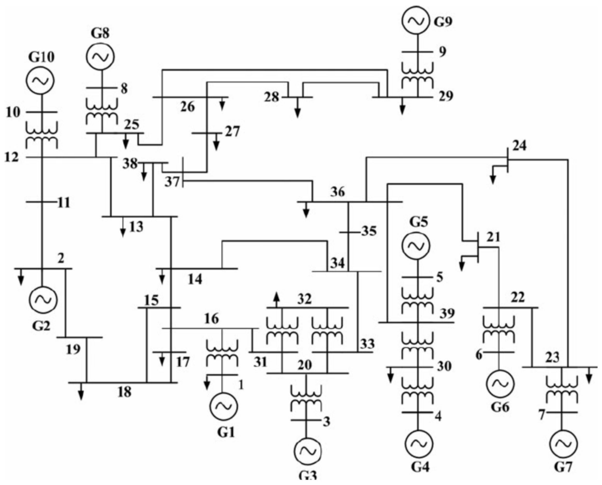

Figure 2: The diagram of the 39-node system

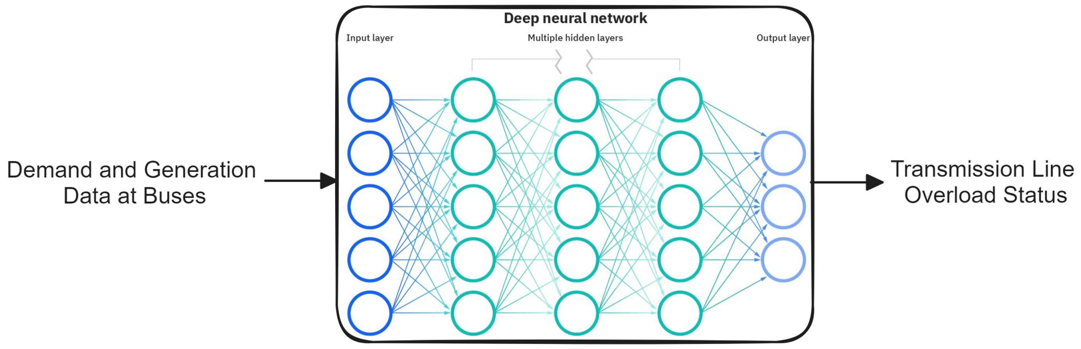
Deep neural network
Input layer
Multiple hidden layers
Output layer
Demand and Generation Data at Buses
Transmission Line Overload Status

Figure 3: Machine-learning-based prediction of transmission line overload

Utilizing Machine Learning to Enhance System Security against Data Manipulation Threats

Following the recognition of potential vulnerabilities in the data of transmission line overload statuses, in this project we aim to leverage machine learning as a tool to mitigate these risks. The objective is to develop a machine learning model that can analyze the trusted demand and generation data from the buses and use this information to reliably predict actual transmission line overload scenarios (see Figure 3). By doing so, the model will serve as a safeguard, identifying discrepancies or anomalies in branch overload status reports that could indicate cyber intrusions or attacks.

1 Part 1: Data Exploration [20 Points]

In the first step, let's explore the given datasets to get familiar with it. First, download the 39-bus system measurement data from this link. The `39-bus-measurements` directory you downloaded contains the following data:

1. Demand and generation data in `train_features.csv` and `test_public_features.csv`.

This data contains active power demands in megawatt MW (from `Pd_bus1` to `Pd_bus39`), reactive power demands in megaVar MVAR (from `Qd_bus1` to `Qd_bus39`), active power generations in megawatt MW (from `Pg_gen1` to `Pg_gen10`), and finally reactive power generations in megaVar MVAR (from `Qg_gen1` to `Qg_gen10`).

2. Transmission line overload status data in `train_labels.csv` and `test_public_labels.csv`. This data contains labels indicating whether each branch (from `is_branch1_Overloaded` to `is_branch46_Overloaded`) is overloaded (1) or not (0), with each row representing a different data point.

We will utilize this data in later sections for training and testing our machine learning model. Before we move on, it is important to explore the data. In this section of the project, you will be answering questions on Canvas: Please see the Canvas "Mini Project 4 - QA" Assignment under "Mini Projects 3 and 4" for the questions to answer. You can submit the quiz multiple times so feel free to look at the questions at any time, but ***please note that only the latest submission will be graded and considered***.

NOTE: Before you jump into the coding part, please check out Section 5 on how you can setup the development/testing environment for this project. This is necessary to ensure we can reproduce your results and maintain compatibility across all submissions. Moreover, if you are new to Python, machine learning, and jupyter notebook, it can help you setup the environment very quickly. In Part 1, it is recommended to use jupyter notebook for quick development/testing while Part 2 requires you to complete the code snippets to develop an acceptable solution.

1. Using the pandas library in Python, load the file `train_features.csv` into a DataFrame named `df_train_features`. Then, answer the following questions:

   (a) What is the shape of `df_train_features`? Specifically, how many data points (rows) and features (columns) does this dataset contain? You can find this out using `df_train_features.shape`. [3 Points]

   (b) How many columns (features) in the dataset are entirely zeros? To identify these, you may utilize the pandas `describe()` method to review the data's summary statistics within the DataFrame. [4 Points]

   (c) How many columns (features) in the dataset have constant values (excluding any columns that are entirely zeros)? [4 Points]

   (d) Which column (feature) in the dataset exhibits the largest variation? To determine this, use the standard deviation (std) as your measure of variation. [4 Points]

2. Using the pandas library in Python, load the file `train_labels.csv` into a DataFrame named `df_train_labels`. Then, answer the following questions:

   (a) Identify the branch with the largest overload frequency. This means the branch that has the highest proportion of its data points marked as overloaded (1). Similarly, identify the branch with the smallest overload frequency. This indicates the branch that has the lowest proportion of its data points marked as overloaded (1). This exercise will help you understand data balance and the distribution of overloaded conditions across various branches. [5 Points]

2 Part 2: Completing the Provided Python Code Template

In this section of the project, we aim to construct a machine learning model using the PyTorch library to predict the overload status of power system branches, a crucial task for validating the genuine behavior of our cyber-physical system, enabling the identification of discrepancies that might arise from cyber-physical attacks. To streamline the development process, a Python code template is provided here. First, clone the repository to your local machine or download it as a ZIP file and unzip it to access the code. The extracted folder, named `mp4-machine-learning-template`, includes the following directories:

- `src`: This directory hosts the Python source code templates. You will need to review and complete these files as outlined in subsequent sections. **Ensure you retain these files within this directory throughout the development process to guarantee the entire code functions correctly. Avoid moving the files out of this directory. Instead, modify and finalize them within this folder.**
- `data`: This directory contains the data. Specifically, the four CSV data files reside in the relative path `data/39-bus-measurements/`.
- `model`: This folder will be empty upon downloading the code template. After training your model, the model and parameter files should be saved here.

Please maintain the specified folder structure and refrain from renaming or reorganizing the folders and files described. This template establishes a structured approach for your machine learning model, incorporating pre-defined classes and functions specifically designed for this project's goals. **Your primary responsibility is to complete the template by filling in the missing sections as directed in the following paragraphs.** It is vital to adhere to the template's structure and utilize the specified libraries to ensure uniformity across all submissions, focusing on applying machine learning techniques in the context of cyber-physical systems security. In our supervised machine learning process, the workflow is divided into two main stages: training and testing.

1. **Training**: During the training stage, we teach the machine learning model to identify patterns in the data. This is done by running the `train.py` script. See Figure 4.
2. **Testing**: After the training stage is complete, we evaluate the model's ability to learn effectively. This evaluation takes place in the testing stage, which is carried out by executing the `test.py` script. See Figure 5.

The code template in the `src` directory includes several key components:

- **Helper Script Files (DO NOT CHANGE):**
  - `train.py`: Manages the model training process. See Figure 4.
  - `test.py`: Evaluates the model's performance on test data. See Figure 5.
- **DataReader:** Handles reading and preprocessing of input data, including tasks like loading data from CSV files, normalizing the data, and formatting it for neural network processing.

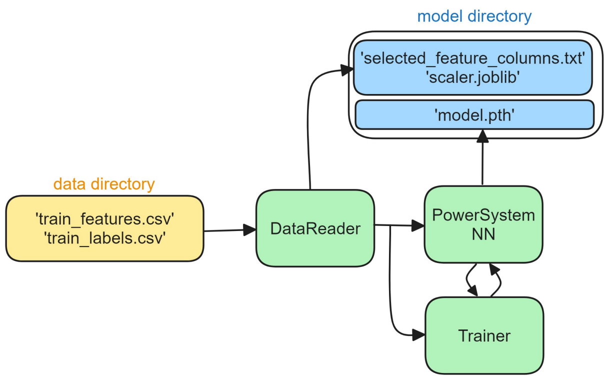

Figure 4: Training workflow

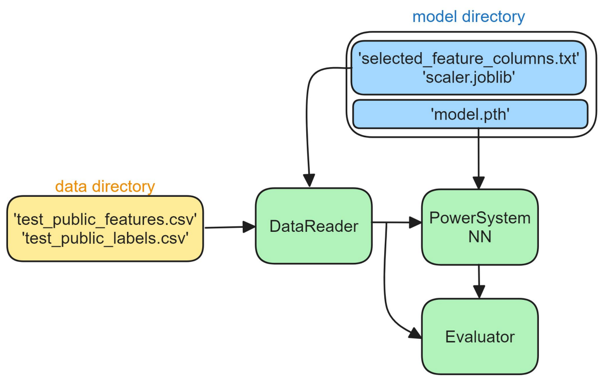

Figure 5: Testing workflow

- **PowerSystemNN:** Defines the neural network architecture tailored for analyzing power system data, including specifying layers and structural elements of the model. See Figure 3.
- **Trainer:** Oversees the neural network's training process, including setting up the loss function and optimizer and executing the training routine with the preprocessed data.
- **Evaluator:** Utilized for evaluating the trained model's performance, calculating metrics such as accuracy, precision, and recall to assess the model's predictive accuracy regarding power system branch overloads. **(DO NOT CHANGE)**

These components collectively provide a robust framework for your project, guiding you through a structured approach to building and evaluating your machine learning model. Your challenge is to complete and potentially enhance these components, implementing the necessary logic to achieve the project's objectives. The following detailed steps will help you navigate through the process efficiently. To begin, please download the template code from the link above and follow the outlined steps:

- Note: In some of the subsequent steps, such as identifying important features in Step 3 or designing the neural network in Step 4, you should begin with an initial working solution. Once you have completed the entire training and testing workflow, you can return to these sections for fine-tuning.

2.1 Step 1: Review `train.py` [0 Points]

The `train.py` script is complete and ready-to-use. This file serves as the core of the machine learning training process for predicting the overload status of branches in our electric power system using neural networks. It outlines the main steps required for training a model, including loading and preprocessing data, initializing and training the neural network, and saving the trained model along with its scaler and selected feature columns for future use. See Figure 4. **You are expected to review and understand this script to grasp the overall structure and flow of the model training pipeline without needing to modify it.**

2.2 Step 2: Review `test.py` [0 Points]

The `test.py` script is also complete and ready-to-use. This file is designed to evaluate the performance of a neural network model on a test dataset. It serves as the counterpart to the training process outlined in `test.py`, focusing on model evaluation rather than model training. See Figure 5. **You are expected to review and understand this script to grasp the overall structure and flow of the model testing pipeline without needing to modify it.**

2.3 Step 3: Complete `data_reader.py` [5 Points]

The `data_reader.py` script is a critical component of the machine learning pipeline, designed to handle data loading, preprocessing, normalization, and conversion to the appropriate formats for training neural network models. **You are tasked with completing the `_find_important_features` method within this class.**

Note that selecting the most important features is a crucial step in machine learning because it can significantly improve model performance by reducing overfitting, decreasing training time, and potentially enhancing prediction accuracy. The purpose of the `_find_important_features` method is to analyze the dataset and identify the features that contribute the most to predicting the target variable, in this case, the overload status of electric power system branches. This involves determining which features have the most significant impact on the model's predictions and selecting them for use in training the model. Note that you should begin with an initial working solution (such as setting `selected_columns` equal to `ALL_FEATURE_LIST`). Once you have completed the entire training and testing workflow, you can return to this section for fine-tuning. In the code you submit for this section, please include the rationale behind your selection of important features.

```python
def _find_important_features(self, df_feature: pd.DataFrame, df_labels: pd.DataFrame)
    -> list:
    selected_columns = [] # COMPLETE HERE
    return selected_columns
```

Note: Please DO NOT alter any other class methods or features in `data_reader.py`. Only complete or modify the `_find_important_features` method.

2.4 Step 4: Complete `power_system_nn.py` [20 Points]

For the `power_system_nn.py` script, you are tasked with designing and implementing the neural network architecture for predicting the overload status of branches in an electric power system using PyTorch (see Figure 3). This involves specifying the structure of the neural network, including the number of layers, the size of each layer (i.e., the number of neurons), and the activation functions to be used. The neural network architecture provided here serves as an example; students are encouraged to experiment with different configurations to optimize model performance. Here are more hints:

1. **Design the Neural Network Architecture:** You will define the neural network's architecture in the `__init__` method. This includes deciding on the number of layers, the number of neurons in each layer, and the type of each layer (e.g., fully connected layers, convolutional layers for other types of data, etc.). You should also choose appropriate activation functions for each layer to introduce non-linearity into the model.

2. **Implement the Forward Pass:** In the forward method, you will specify how the data flows through the network. This involves applying the layers and activation functions defined in the `__init__` method to the input tensor `x` and returning the output tensor.

3. **Activation Functions:** Consider the role of different activation functions (e.g., ReLU, Sigmoid, Tanh) and where they might be most effectively applied within your network to model complex relationships in the data.

4. **Experimentation:** Experiment with different network configurations and parameters (e.g., different numbers of layers, different numbers of neurons in each layer, different activation functions) to find a setup that works well for the task.

5. The example layer and activation function definitions in the comments are provided as hints. Replace `number_of_neurons` and other placeholders with specific values based on your design decisions.

6. You should begin with an initial working solution (such as 1-2 hidden layers and a reasonable number of neurons in each layer, etc.). Once you have completed the entire training and testing workflow, you can return to this section for fine-tuning.

7. In this link, there is an example of a neural network that includes the initialization and forward pass methods.

8. Please see the provided `power_system_nn.py` file for more hints/comments.

2.5 Step 5: Complete `trainer.py` [20 Points]

In the `trainer.py` script, you are tasked with implementing the functionality required to train a neural network model, including initializing the training components and executing the training loop. This involves setting up the optimizer and loss function in the `__init__` method and managing the data loading, model training iterations, and optimization steps in the `train_model` method. Here are more hints:

1. **Initialize Training Components:** In the `__init__` method, you must initialize the model, loss function, and optimizer. This involves specifying the type of loss function suitable for the task (e.g., Binary Cross-Entropy for binary classification tasks) and choosing an optimizer (e.g., Adam) with an appropriate learning rate.

2. **Implement the Training Loop:** The `train_model` method is responsible for organizing the training process. This includes setting up a DataLoader for batching the training data, iterating over the dataset for a defined number of epochs, performing forward and backward passes through the model, computing the loss, and updating the model's weights.
3. In this link, there is an example of setting up the optimizer and loss function.
4. Please see the provided `trainer.py` file for more hints/comments.

2.6 Step 6: Review `evaluator.py` [0 Points]

The `evaluator.py` script, which is complete and ready-to-use, evaluates the performance of your trained neural network model on test datasets. Specifically, the `evaluate` method within the Evaluator class will assess the model's prediction accuracy, precision, and recall, leveraging the `sklearn.metrics` library for these calculations. **You are expected to review and understand this script without needing to modify it.** Here are more details:

1. **The Evaluate Method:** In the evaluate method, we predict outcomes using the trained model on a dataset provided by a DataReader instance. The method then calculates and reports the accuracy, precision, recall, and F1 score for the predictions.
2. **Using of sklearn.metrics:** The `accuracy_score`, `precision_score`, `recall_score`, and `f1_score` functions from `sklearn.metrics` are used to calculate the respective metrics. These metrics provide insights into the model's performance, with accuracy indicating the overall correctness of predictions, precision showing the correctness of positive predictions, and recall reflecting the model's ability to identify all actual positives:
    - Accuracy: Accuracy measures the proportion of total correct predictions (both true positives and true negatives) out of all predictions. Generally, a higher accuracy is better, but it can be misleading in the case of imbalanced datasets where one class dominates.
    - Precision: Precision measures the proportion of true positives out of all positive predictions. It focuses on the purity of positive predictions. A higher precision is typically better, especially in scenarios where false positives are costly or undesirable. However, extremely high precision can sometimes occur at the expense of recall, especially if the model becomes too conservative in predicting the positive class.
    - Recall: Recall (or sensitivity) measures the proportion of actual positives that are correctly identified. It focuses on the model's ability to capture all positive instances. Higher recall is desirable in scenarios where missing a positive instance (false negatives) is critical. However, very high recall can sometimes come at the expense of precision, if the model predicts too many positives, including false ones.
    - F1 Score: The F1 Score is the harmonic mean of precision and recall, providing a single metric that balances both. An F1 Score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is particularly useful in scenarios where there is an imbalance in the distribution of the classes or when one cares equally about precision and recall.
3. **Calculating Metrics for Each Branch and on Average:** We calculate these metrics for each power system branch individually and also compute average metrics across all branches to get a holistic view of the model's performance.

3 Deliverable and Submission Instructions

Create a zip file named <First Name>-<Last Name>-mp4.zip (e.g., Tohid-Shekari-mp4.zip), that includes all your files and submit it on Canvas. If you make multiple submissions to Canvas, a number will be appended to your filename. This is inserted by Canvas and will not result in any penalty.

- ***Note 1: Please ensure you use the virtual environment with the specified package versions in the requirements.txt for training and generating the submission files (see Section 5.2). This is necessary to ensure we can reproduce your results and maintain compatibility across all submissions.***
- ***Note 2: Failure to follow the submission and naming instructions will cause 20% points loss.***

```
<First Name>-<Last Name>-mp4.zip
|-- src
   |-- data_reader.py
   |-- evaluator.py
   |-- power_system_nn.py
   |-- test.py
   |-- train.py
   |-- trainer.py
|-- model
   |-- model.pth
   |-- scaler.joblib
   |-- selected_feature_columns.txt
   |-- public_points.txt
```

4 Model Evaluation [35 Points]

Your model's performance will be assessed using two test datasets: a public dataset that has been shared with you, and a private dataset that has not been shared and will be used to evaluate the model's generalizability. The overall performance of your model will be determined by the combined performance on these two datasets, calculated as follows:

$$\begin{aligned}\text{Total Points} = &\left(P^{\text{public}} \times \frac{(X^{\text{public}} + Y^{\text{public}})}{2} \times Z^{\text{public}}\right)\\ &+ \left(P^{\text{private}} \times \frac{(X^{\text{private}} + Y^{\text{private}})}{2} \times Z^{\text{private}}\right)\end{aligned} \tag{1}$$

where $P^{\text{public}} = 15$ and $P^{\text{private}} = 20$. Additionally, X, Y, and Z will be calculated according to Tables 1, 2, and 3. This formula exists in the `test.py` script and it will automatically calculate your score on the public test dataset.

Table 1: Lookup table for calculation of X

Average Accuracy Across All Branches	X
$[95\%, 100\%]$	1
$[90\%, 95\%)$	0.9
$[85\%, 90\%)$	0.7
Below 85%	0

Table 2: Lookup table for calculation of Y

Average F1 Score Across All Branches	Y
$[90\%, 100\%]$	1
$[85\%, 90\%)$	0.9
$[80\%, 85\%)$	0.7
Below 80%	0

Table 3: Lookup table for calculation of Z

Number of Feature Columns Used	Z
$\{1, \cdots, 53\}$	1
$\{54, \cdots, 64\}$	0.9
$\{65, \cdots, 80\}$	0.7
Above 80	0

5 Appendix: Installation and Getting Started with Python, PyTorch, scikit-learn, and Jupyter Notebook

5.1 Installing Python

For this course project, you will need Python 3.11, which is recommended for optimal performance and compatibility. There are plenty of resources you can find publicly on how you can install Python in your Windows, MacOS, or Linux machine.

5.2 Setup the Project Environment

Using a virtual environment helps isolate dependencies and avoid conflicts with other projects, ensuring that your code runs smoothly and consistently on different systems.

Please ensure you use the virtual environment described in this section with the specified package versions in the requirements.txt for training and generating the submission files. This is necessary to ensure we can reproduce your results and maintain compatibility across all submissions.

Once you successfully installed Python 3.11 in your machine, clone this github repo which includes all the required datasets and code skeletons in parts 1 and 2 of the project. Navigate to the source directory with terminal and run this command to create a virtual environment for the project:

```
python3 -m venv .venv
```

Before you can start using the virtual environment, you need to activate it. Activation is necessary because it temporarily modifies the PATH environment variable to include the scripts in the virtual environment's bin (or Scripts on Windows) directory. To activate the virtual environment, run (first one in Windows, the second one in Mac/Linux):

```
.\.venv\Scripts\activate
```

```
source .venv/bin/activate
```

With the virtual environment activated, install the dependencies listed in your requirements.txt file by running:

```
pip3 install -r requirements.txt
```

5.2.1 Optional: Setting Up Jupyter Notebook for Facilitating the Data Exploration in Part 1

If you need jupyter notebook, first run this command to create a kernel based on the virtual environment and then use the next command to launch an instance of it.

```
python -m ipykernel install --user --name=mp4env --display-name="mp4env"
```

```
jupyter notebook
```

6 Appendix: Distribution of Points

The distribution of points for this project is illustrated in Figure 6.

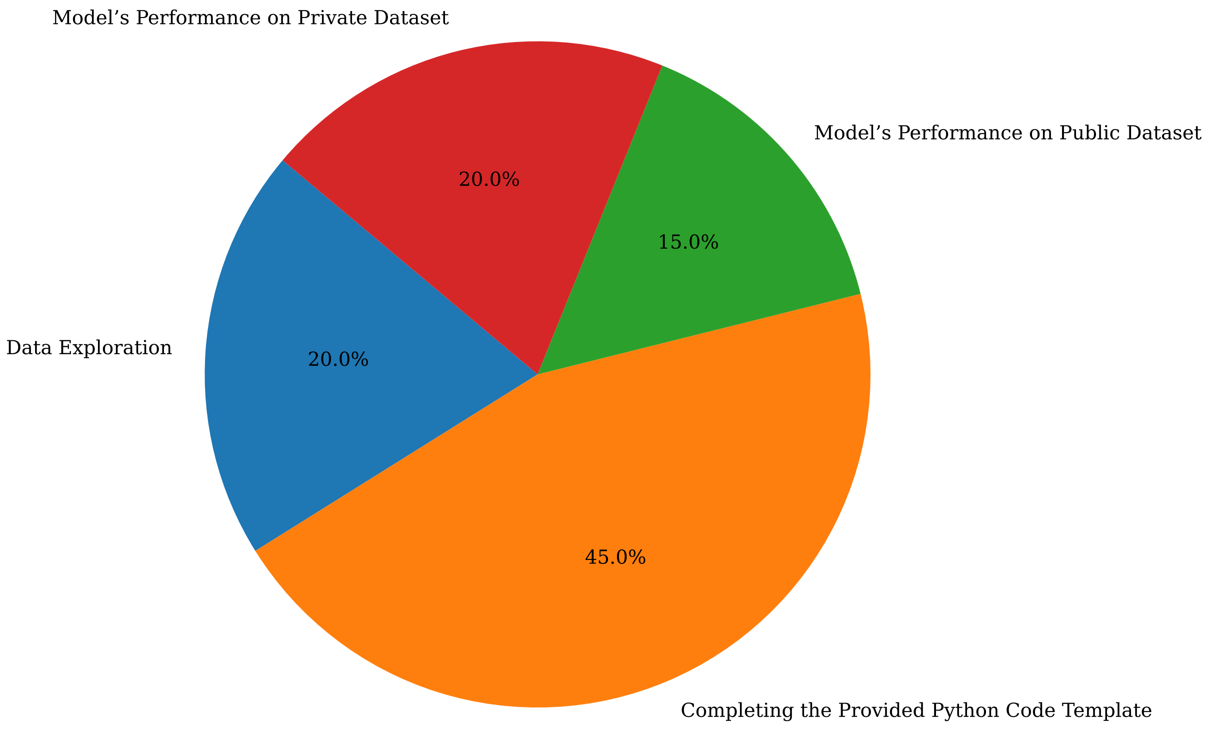

Figure 6: Distribution of points

Resources

- PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`
- scikit-learn Documentation: `https://scikit-learn.org/stable/`
- Pandas Documentation: `https://pandas.pydata.org/docs/`
- Jupyter Notebook Documentation: `https://docs.jupyter.org/en/latest/`