University of Delaware

# Engineering Machine Learning Systems
# Checkpoint 1

Saeid Rajabi
Department Of Electrical and Computer Engineering
College of Engineering

# Contents

**1 Model Design** **2**
1.1 Choice of Models . . . 2
1.2 Architecture . . . 2
1.2.1 Base Models . . . 2
1.2.2 Meta-Learner (MLP) . . . 3
1.3 Implementation Details . . . 3

**2 Training Process** **5**
2.1 Hyperparameter Tuning . . . 5
2.1.1 Base Models . . . 5
2.1.2 Meta-Learner . . . 5
2.2 Training . . . 5
2.2.1 Base Models . . . 5
2.2.2 Meta-Learner . . . 6
2.3 Addressing Overfitting and Underfitting . . . 6

**3 Preliminary Results** **7**
3.0.1 Base Models . . . 7
3.0.2 Meta-Learner (MLP) . . . 7
3.1 Analysis . . . 7

**4 Next Steps** **10**

**5 Multi-class classification** **11**
5.1 Choice of Models . . . 11

**6 Training Process** **11**
6.1 Data Preparation . . . 11
6.2 Data Splitting . . . 11
6.3 Hyperparameter Tuning . . . 11
6.4 Training and Evaluation . . . 11

**7 Preliminary Results** **12**
7.1 Base Model Performance . . . 12
7.1.1 Logistic Regression . . . 12
7.2 Meta-Learner Training . . . 12
7.2.1 Decision Tree . . . 13
7.2.2 Random Forest . . . 13

**8 Analysis** **13**
8.1 Meta-Learner Performance Analysis . . . 14
8.2 Comparison with Base Models . . . 14

**9 Next Steps** **14**

# 1 Model Design

## 1.1 Choice of Models

To address the binary classification problem, I selected an ensemble of three base models and a meta-learner:

1. **Logistic Regression (LR):**
   - I chose it for its simplicity. The logistic regression serves as a strong baseline model. Its ability to model linear relationships helps us assess the fundamental separability of classes within our dataset.
2. **Decision Tree (DT):**
   - Decision Trees can capture non-linear relationships and are useful for understanding feature importance in the tree structure. I adjusted hyper-parameters (hyper-parameter tuning) like tree depth; I tried to reduce overfitting and improve generalization.
3. **Random Forest (RF):**
   - This algorithm is an ensemble method of multiple decision trees. RFs are strong against overfitting and can handle noise and feature redundancy. They improve accuracy by aggregating the predictions of various trees.
4. **Meta-Learner (Multilayer Perceptron - MLP):**
   - An MLP is employed as the meta-learner to aggregate the predictions from the base models. Its capability to learn complex non-linear patterns across meta-features makes it suitable for capturing the underlying relationships missed by base models.

## 1.2 Architecture

### 1.2.1 Base Models

- **Logistic Regression:**
  - Regularization ($C$ or $\frac{1}{\lambda}$): 10.0
  - Maximum iterations: 20
- **Decision Tree:**
  - Maximum depth: 8
- **Random Forest:**
  - Number of trees: 50
  - Maximum depth: 10

#### 1.2.2 Meta-Learner (MLP)

- **Input Layer:**
  - Size: 6 nodes (each base model provides two probability estimates: one for each class)
- **Hidden Layers:**
  - Layer 1: 32 nodes, ReLU activation
  - Layer 2: 16 nodes, ReLU activation
  - Layer 3: 8 nodes, ReLU activation
- **Output Layer:**
  - 2 nodes (for binary classification), softmax activation for probabilistic outputs
- **Regularization:**
  - Dropout rate: 0.2
  - L2 regularization applied to weights (i.e., wight decay)

### 1.3 Implementation Details

- **Libraries and Frameworks:**
  - **Scikit-learn:** Used for implementing the base models, data preprocessing, and evaluation metrics.
  - **PyTorch:** Utilized for building and training the MLP meta-learner.
- **Challenges Faced:**
  - *Computational Resources:* Training on a large dataset was computationally intensive, necessitating optimization techniques like mini-batch processing and efficient data loaders. For this preliminary result, I used 20% of the balanced dataset or 947242 instances.
  - *Integration of Models:* Integrating outputs from scikit-learn models with PyTorch presented several challenges. The base models, implemented using scikit-learn, produce output probabilities as NumPy arrays, while the PyTorch meta-learner requires input data in the form of tensors. Ensuring data compatibility involved:
    - **Data Conversion:** I implemented an convertor function that converts NumPy arrays to PyTorch tensors, (e.g., float32 vs. float64) to prevent type mismatch errors.
    - **Dimensional Consistency:** I reshaped input data to ensure that it meets the expectations of the meta-learner inputs for the dimensions. (i.e., torch.tensor(X_train_meta, dtype=torch.float32))

Despite these integration efforts, the base models initially exhibited high prediction accuracies, which limited the opportunity for the meta-learner to demonstrate significant improvements. To address this and further challenge the robustness of our ensemble approach, I introduced a controllable **corruption rate** to the outputs of the base models:

* **Corruption Implementation:** I added a controllable noise to the probability outputs of the base models by flipping a certain percentage of the predictions or adding random Gaussian noise to the probabilities. The reason was that simulated real-world scenarios where base models might produce unreliable predictions due to data noise. I observed that even after corrupting the outputs of the base models, the MLP meta-learner maintained high accuracy levels. This indicates that the meta-learner effectively learned to correct or mitigate the errors introduced in the base models' predictions.

# 2 Training Process

- **Training and Testing Split:**
  - I split the dataset into 80/20 fractions and used 80% for training and 20% for testing. For the validation during training, I implemented 3-fold cross-validation within the training set for hyper-parameter tuning and model evaluation.

## 2.1 Hyperparameter Tuning

### 2.1.1 Base Models

- **Parameters:**
  - **Logistic Regression:**
    - $C$ values: [0.1, 1.0, 10.0]
  - **Decision Tree:**
    - Max depths: [5, 8, 12]
  - **Random Forest:**
    - Number of trees: [50, 100]
    - Max depths: [10, 15]
- **Optimization Method:**
  - Employed grid search with cross-validation to identify the optimal hyperparameters for each base model.

### 2.1.2 Meta-Learner

- **Hyperparameters:**
  - Learning rates: [0.001, 0.0001]
  - Batch sizes: [32, 64]
  - Optimizers: [Adam]

## 2.2 Training

### 2.2.1 Base Models

- **Process:**
  - I trained each model using the training set with the optimal hyperparameters. Then, I collected probability estimates for each class, which serve as input tensors (meta-features) for the meta-learner.

#### 2.2.2 Meta-Learner

- **Process:**
  - **Input Preparation:** I combined the probability outputs from the base models to form a new feature set as an tensor format.
  - **Training Loop:**
    - Number of epochs: 50
    - Loss function: Cross-entropy loss
    - Optimizer: Adam optimizer with weight decay for L2 regularization
    - Batch size: 64
  - **Regularization Techniques:**
    - I applied dropout layers with a rate of 0.2 after each hidden layer. Also, I implemented L2 regularization to penalize large weights.

### 2.3 Addressing Overfitting and Underfitting

- In the 3-fold cross-validation provided an estimate of model performance and reduced variance during training.
- **Regularization:**
  - I used dropout and L2 regularization in the MLP to prevent overfitting.

# 3 Preliminary Results

### 3.0.1 Base Models

The base models were evaluated before and after introducing corruption to their outputs.

**Before Corruption:**

- All base models (**Logistic Regression**, **Decision Tree**, **Random Forest**) achieved near-perfect performance:
  - **Accuracy:** Approximately 100%.
  - **Precision, Recall, F1-Score:** All close to 1.00 for both classes.

**After Corruption:**

- **Corruption Rates:** Varied across models and folds, ranging from approximately 31% to 49%.
- **Performance Degradation:** The accuracy of base models dropped significantly:
  - **Logistic Regression:** Accuracy reduced to between 52% and 68%.
  - **Decision Tree:** Accuracy reduced to between 50% and 64%.
  - **Random Forest:** Accuracy reduced to between 55% and 68%.
- **Classification Reports:** Showed that precision, recall, and F1-scores decreased correspondingly, indicating that the models' predictions were close to random guessing.

### 3.0.2 Meta-Learner (MLP)

Despite the corrupted inputs from the base models, the MLP meta-learner achieved near-perfect performance:

- **Training Loss:** Decreased steadily over 50 epochs, from 0.6894 to 0.4038, indicating effective learning.
- **Accuracy:** Achieved 100% accuracy on the test data.
- **Classification Report:** Precision, recall, and F1-scores were all 1.00 for both classes.

## 3.1 Analysis

- **Impact of Corruption on Base Models:**
  - Introducing corruption significantly degraded the performance of the base models. The models' accuracies dropped to near-chance levels, simulating real-world scenarios with unreliable predictions due to noise conditions.
- **Meta-Learner Robustness:**
  - The meta-learner maintained high performance despite the corrupted inputs.

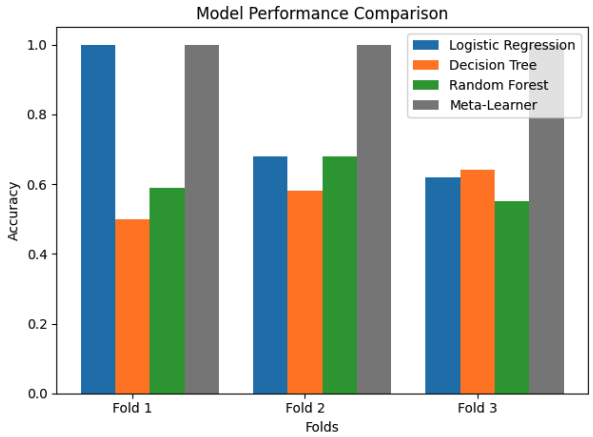


Figure 1: Each model's accuracy in each fold of the training

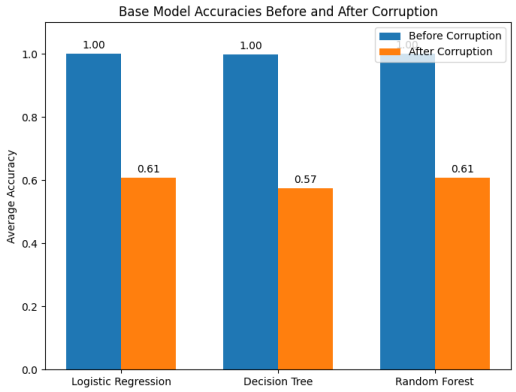


Figure 2: Comparision between the accuracies of the base models before and after the noise adding

  - This suggests that the meta-learner learned to mitigate the errors introduced by the base models and the consistent decrease in training loss over epochs indicates a good generalization from the corrupted data.

- **Overfitting Assessment:**

  - Despite achieving 100% accuracy, overfitting did not happen on the dataset subset that I selected since the model performed equally well on unseen test data.

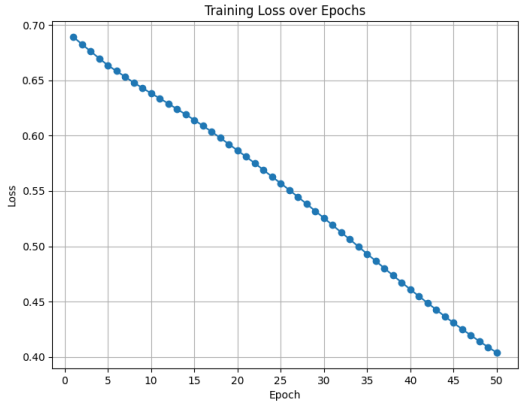


Figure 3: Training loss over the epochs

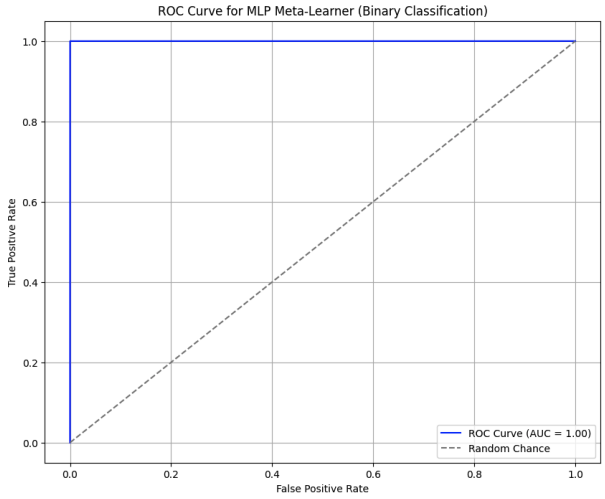


Figure 4: ROC curve of MLP model

MLP Meta-Learner Performance:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 199921 |
| 1 | 1.00 | 1.00 | 1.00 | 199920 |
| accuracy | | | 1.00 | 399841 |
| macro avg | 1.00 | 1.00 | 1.00 | 399841 |
| weighted avg | 1.00 | 1.00 | 1.00 | 399841 |

Figure 5: Performance of the MLP model

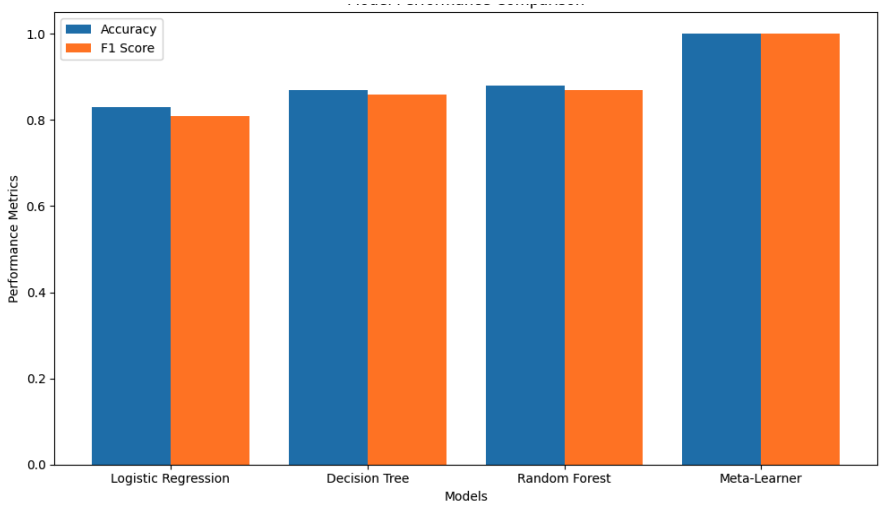


Figure 6: Comparison of each model accuracy and F1-score

## 4 Next Steps

As of the next steps, I am going to do some experiments on the larger portion of the dataset to see if the corruption function is required or not. For the model optimization, I am going to experiment with alternative architectures for the meta-learner, including deeper networks or ensemble MLPs. Also, hyperparameter tuning for the meta-learner with Bayesian optimization will be done, plus for doing some experiments on SGD.

# 5 Multi-class classification

In this section of the report, there is a short report for the second part of the project on the multi-class classification.

## 5.1 Choice of Models

To address the multiclass classification problem, I selected the following base models: The reasons for selecting these models are the same as for the binary classification.

1. **Logistic Regression (LR)**
2. **Decision Tree (DT)**
3. **Random Forest (RF)**

# 6 Training Process

## 6.1 Data Preparation

I encoded the categorical attack class labels into numerical labels using label encoding. Then, I standardized the features using standard scaling to ensure all features contributed equally to the model training. Note to be mentioned that I used 10% of the data.

## 6.2 Data Splitting

I split the data into training and testing sets using an 80/20 split, same as the binary section.

## 6.3 Hyperparameter Tuning

- **Logistic Regression:**
  - Regularization: L2
  - Inverse of regularization strength ($C$): 1.0
  - Maximum iterations: 100
- **Decision Tree:**
  - Maximum depth: 10
- **Random Forest:**
  - Number of trees: 50
  - Maximum depth: 10

## 6.4 Training and Evaluation

I followed the same approach, 3-fold cross-validation to train and evaluate the models.

# 7 Preliminary Results

## 7.1 Base Model Performance

### 7.1.1 Logistic Regression

**Observations:**

- **Accuracy:** Approximately 82% to 83% across folds.
- **Strengths:** Performed well on classes 0, 3, 7, 8, 9, and 10 with high precision and recall.
- **Weaknesses:** Struggled with class 4, where precision and recall were near zero, indicating difficulty in correctly identifying this class.

## 7.2 Meta-Learner Training

I implemented a similar MLP as a meta-learner to aggregate the predictions from the base models. The MLP architecture consisted of:

- **Input Layer:** Size equal to the combined output probabilities of the base models.
- **Hidden Layers:**
  - Layer 1: 64 neurons, ReLU activation
  - Layer 2: 32 neurons, ReLU activation
  - Layer 3: 16 neurons, ReLU activation
- **Output Layer:** 11 neurons (one for each class), softmax activation for probabilistic outputs.

**Training Details:**

- **Loss Function:** Cross-entropy loss
- **Optimizer:** Adam optimizer with a learning rate of 0.001 and weight decay of $1 \times 10^{-5}$ for L2 regularization.
- **Batch Size:** 64
- **Number of Epochs:** 200

**Training Process:**
I trained the MLP on the meta-features (probability outputs) collected from the base models. The training loss decreased steadily over epochs, indicating effective learning despite the challenges posed by class imbalance and the complexity of the multiclass classification task.

### 7.2.1 Decision Tree

**Observations:**

- **Accuracy:** Approximately 87% to 88% across folds.
- **Strengths:** Improved performance over Logistic Regression, with high precision and recall for most classes.
- **Weaknesses:** Similar to LR, struggled significantly with class 4.

### 7.2.2 Random Forest

**Observations:**

- **Accuracy:** Approximately 88% across folds.
- **Strengths:** Achieved high precision and recall for most classes.
- **Weaknesses:** Despite overall strong performance, class 4 remained problematic, with near-zero recall.

**Classification Report:**

The MLP meta-learner achieved the following performance on the test set:

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 81,464 |
| 1 | 1.00 | 1.00 | 1.00 | 76,784 |
| 2 | 1.00 | 1.00 | 1.00 | 54,375 |
| 3 | 1.00 | 1.00 | 1.00 | 75,323 |
| 4 | 0.74 | 0.00 | 0.00 | 110,874 |
| 5 | 0.51 | 1.00 | 0.68 | 115,693 |
| 6 | 1.00 | 1.00 | 1.00 | 99,279 |
| 7 | 1.00 | 1.00 | 1.00 | 76,113 |
| 8 | 1.00 | 1.00 | 1.00 | 78,619 |
| 9 | 1.00 | 1.00 | 1.00 | 92,477 |
| 10 | 1.00 | 1.00 | 1.00 | 86,241 |
| **Accuracy** | 0.88 (on 947,242 samples) | | | |
| **Macro Avg** | 0.93 | 0.91 | 0.88 | |
| **Weighted Avg** | 0.91 | 0.88 | 0.84 | |

Table 1: Classification Report for MLP Meta-Learner

## 8 Analysis

I observed that class 4 had a significant number of samples but showed poor performance across all models. The zero recall indicated that the models failed to correctly predict any instances

of class 4. This suggests that the features may not provide enough discriminatory power for this class, or there may be an issue with the data related to this class.

### 8.1 Meta-Learner Performance Analysis

The MLP meta-learner showed significant improvement in classifying several classes compared to the base models. It achieved perfect scores on most classes, demonstrating the effectiveness of aggregating base model predictions.

However, class 4 remained a challenge:

- **Class 4 Issues:** The meta-learner achieved a precision of 0.74 but a recall of 0.00 for class 4. This indicates that while some instances were predicted as class 4, none of the actual class 4 instances were correctly identified.
- **Class 5 Observation:** The model had a precision of 0.51 and a recall of 1.00 for class 5, suggesting that all instances predicted as class 5 were correct, but there were many false positives from other classes misclassified as class 5.

### 8.2 Comparison with Base Models

- The meta-learner maintained the high performance of the base models on classes where they performed well.
- The inability to improve classification on class 4 suggests that the issue may lie in the data itself, rather than the modeling approach.

## 9 Next Steps

- I will investigate additional features or transformations that could help in distinguishing class 4.
- I will increase the maximum iterations for Logistic Regression to ensure convergence.
- I am going into dive deeper into the problem for the class 4, maybe some data preprocessing is required.

## References