Phương pháp lựa chọn đặc trưng dựa trên Thuật toán di truyền

Nhóm 3:

Huỳnh Minh Tân Tiến	21521520
Nguyễn Ngọc Trà My	21520353
Lê Hoàng Oanh	21521253
Bùi Hoàng Trúc Anh	21521817

01

Vấn đề
bài báo

02

Lý thuyết
liên quan

03

Kịch bản
thực nghiệm

04

Kiến trúc
tổng quan

05

Hiện thực
hệ thống

06

Tài liệu
tham khảo

1. Vấn đề bài báo

Nghiên cứu này tập trung vào việc thiết kế một phương pháp lựa chọn đặc trưng trong lĩnh vực bảo mật mạng và phát hiện xâm nhập. Được gọi là **GA-based Feature Selection (GbFS)**.

Nghiên cứu này cũng tập trung vào việc điều chỉnh tham số cho việc lựa chọn đặc trưng dựa trên GA và đề xuất một hàm thể trạng mới.

2. Lý thuyết liên quan

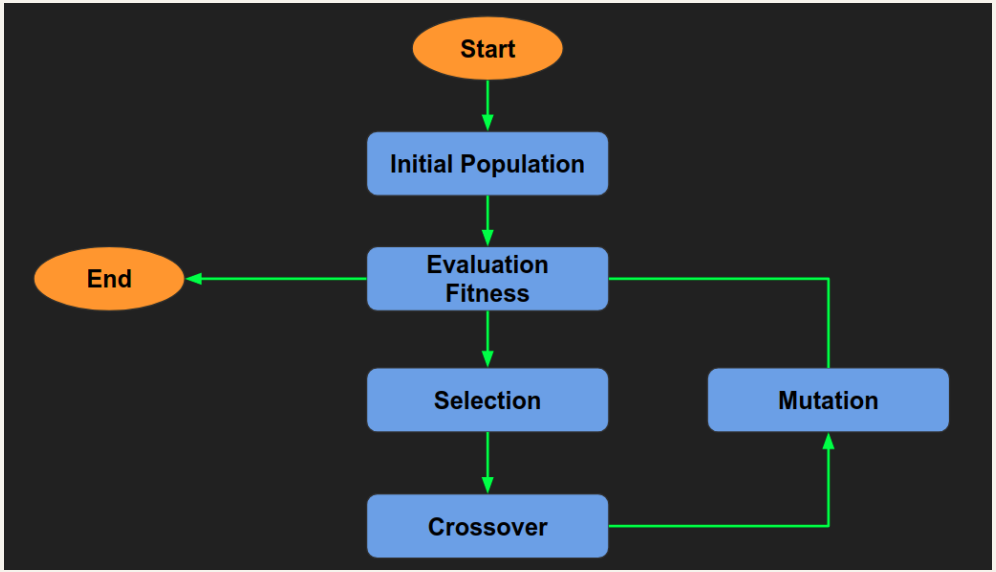
Feature Selection: Quá trình chọn ra một tập hợp con các đặc trưng từ dữ liệu gốc để huấn luyện mô hình máy học, nhằm cải thiện hiệu suất và giảm chi phí tính toán.

Genetic Algorithm (GA): Phương pháp tối ưu hóa dựa trên cơ chế chọn lọc tự nhiên của Darwin, phổ biến trong trí tuệ nhân tạo và học máy.

- Nguyên lý hoạt động
  - Khởi tạo quần thể: Tạo ra quần thể ban đầu.
  - Đánh giá năng lực: Đánh giá mỗi cá thể bằng hàm thích nghi.
  - Chọn lọc: Chọn cá thể tốt để sinh sản.
  - Lai ghép: Tạo ra con cái mới.
  - Đột biến: Thay đổi ngẫu nhiên một số cá thể.

2. Lý thuyết liên quan

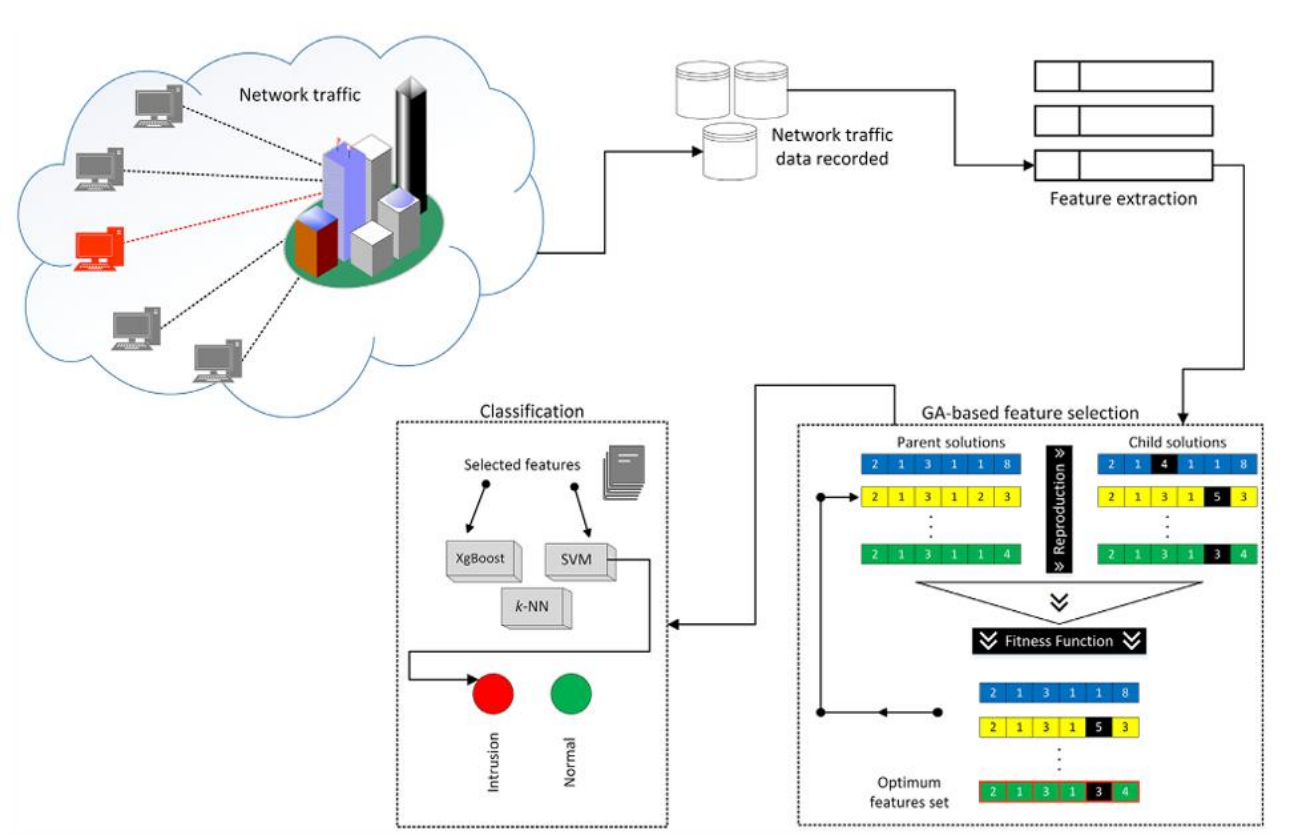
Các bước được lặp lại cho đến khi đạt được một điều kiện dừng nhất định, chẳng hạn như đạt đến số thế hệ tối đa hoặc giá trị fitness mong muốn.



3. Kịch bản thực nghiệm

- Xử lý dữ liệu của ba tập dataset: CIRA-CIC-DOHBrw-2020, UNSW-NB15 và Bot-IoT.
- Huấn luyện các bộ phân loại học máy sử dụng các đặc trưng tối ưu được lựa chọn thông qua mô-đun GA phát triển.
- Đánh giá hiệu suất trên các tập dữ liệu thử nghiệm chuẩn, cụ thể là CIRA-CIC-DOHBrw-2020, UNSW-NB15 và Bot-IoT
- Kiểm tra kỹ thuật đề xuất với ba bộ phân loại, cụ thể là k-Nearest Neighbor (k-NN), Support Vector Machine (SVM), và XgBoost.

4. Kiến trúc tổng quan



5. Hiện thực hệ thống

Tiêu chí đánh giá:

- Accuracy: Đo lường hiệu suất của bộ phân loại, là tỷ lệ giữa số dự đoán đúng và tổng số dự đoán.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- Recall/Sensitivity: Tỷ lệ giữa số dự đoán dương tính đúng và tổng số mẫu dương tính

$$Sensitivity = \frac{TP}{TP + FN}$$

5. Hiện thực hệ thống

Dataset: UNSW-NB15

	Kích thước	Đặc điểm
Train	~30MB: 175000 dòng	Nhận diện đa nhãn: 8 nhãn
Test	~15MB: 80000 dòng	

5. Hiện thực hệ thống

Dataset: BOT-IOT

	Kích thước	Đặc điểm
Train	535217 dòng	Nhận diện đa nhãn:5 nhãn - DDoS - DoS - Reconnaissance - Theft - Normal
Test	133804 dòng	

Dataset Bot-IoT là một bộ dữ liệu được phát triển để nghiên cứu và phát hiện các loại tấn công mạng trong môi trường Internet of Things (IoT).

5. Hiện thực hệ thống

Dataset: CIRA-CIC-DOHBrw-2020

	Kích thước	Đặc điểm
doh	157MB: ~270000 dòng	Nhận diện đa nhãn: 35 nhãn
non_doh	449MB: ~900000 dòng	
benign	10.9MB: ~200000 dòng	
malicious	148MB: ~250000 dòng	

Trong bộ dữ liệu CIRA-CIC-DoHBrw-2020, một phương pháp hai lớp được sử dụng để thu thập lưu lượng DoH (DNS over HTTPS) lành tính và độc hại cùng với lưu lượng không phải DoH.

Ở lớp đầu tiên, lưu lượng thu thập được được phân loại thành DoH và không phải DoH bằng cách sử dụng bộ phân loại dựa trên các đặc trưng thống kê.

Ở lớp thứ hai, lưu lượng DoH được phân loại tiếp thành DoH lành tính và DoH độc hại bằng cách sử dụng bộ phân loại dựa trên chuỗi thời gian.

5. Hiện thực hệ thống

Cài đặt cấu hình hệ thống

- Môi trường phát triển: Google Colab
- Ngôn ngữ lập trình: Python 3
- Cài đặt các thư viện cần thiết

5. Hiện thực hệ thống

Xử lý dữ liệu:

- Sử dụng LabelEncoder để mã hóa các cột dạng chuỗi.

```
# Creating a instance of label Encoder.
le = LabelEncoder()

str_columns = dataset.select_dtypes(include=['object']).columns
for col in str_columns:
    # dataset_l2[col] = le.fit_transform(dataset_l2[col])
    dataset[col] = le.fit_transform(dataset[col].astype(str))
```

- Tỉ lệ hóa dữ liệu sử dụng hàm Min-Max scaling

```
def min_max_scaling(data):
  scaled_dataset = (data-data.min())/(data.max()-data.min())
  return scaled_dataset
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Khởi tạo quần thể

```
def initialize_pop(NUM_FEATURES):
    """
    Khởi tạo quần thể ban đầu.

    Parameters:
    NUM_FEATURES (int): Số lượng đặc trưng.

    Returns:
    list: Danh sách các chromosome trong quần thể ban đầu.
    """
    population = []
    for _ in range(POP_SIZE):
        chromosome = [random.choice([0, 1]) for _ in range(NUM_FEATURES)]  # Sử dụng 0 và 1 để biểu diễn sự có mặt của mỗi đặc trưng
        population.append(chromosome)
    return population
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Fitness function

```python
def fitness_function(data, chromosome):
    """
    Tính toán hiệu suất của một chromosome dựa trên độ chính xác của mô hình học máy.

    Parameters:
    data (DataFrame): Dữ liệu ban đầu.
    chromosome (array-like): Các đặc trưng được chọn từ chromosome.

    Returns:
    float: Độ chính xác của mô hình học máy sử dụng các đặc trưng từ chromosome.
    """
    # Chọn các đặc trưng từ chromosome
    selected_features = data.iloc[:, chromosome]
    X = selected_features.values
    y = data['Label'].values  # Giả sử cột cuối cùng là cột nhãn

    # Chia dữ liệu thành tập huấn luyện và tập kiểm tra
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Huấn luyện mô hình hồi quy logistic
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # Đánh giá độ chính xác của mô hình trên tập kiểm tra
    accuracy = model.score(X_test, y_test)

    return accuracy
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- calculate_fitness_values

```
def calculate_fitness_values(data, population):
    """
    Tính toán giá trị Fitness cho toàn bộ quần thể.

    Parameters:
    data (DataFrame): Dữ liệu gốc.
    population (list): Danh sách các cá thể trong quần thể.

    Returns:
    list: Danh sách các giá trị Fitness tương ứng với mỗi cá thể trong population.
    """
    fitness_values = []
    for chromosome in population:
        fitness_value = fitness_function(data, chromosome)
        fitness_values.append(fitness_value)
    return fitness_values
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Lựa chọn các cá thể cha mẹ

```python
def roulette_wheel_selection(population, fitness_values):
    """
    Lựa chọn các cá thể cha mẹ sử dụng phương pháp bánh xe quay.

    Parameters:
    population (list): Danh sách các cá thể trong quần thể.
    fitness_values (list): Danh sách các giá trị Fitness tương ứng với mỗi cá thể trong population.

    Returns:
    tuple: Một cặp cha mẹ được chọn từ quần thể.
    """
    total_fitness = sum(fitness_values)
    selection_point = random.uniform(0, total_fitness)
    current_sum = 0

    for i, fitness in enumerate(fitness_values):
        current_sum += fitness
        if current_sum >= selection_point:
            parent1 = population[i]
            break

    current_sum = 0
    selection_point = random.uniform(0, total_fitness)
    for i, fitness in enumerate(fitness_values):
        current_sum += fitness
        if current_sum >= selection_point:
            parent2 = population[i]
            break

    return parent1, parent2
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Lai ghép (crossover)

```python
def crossover(parent1, parent2):
    """
    Thực hiện toán tử lai ghép (crossover) giữa hai cá thể cha mẹ.

    Parameters:
    parent1 (list): Cá thể cha mẹ thứ nhất.
    parent2 (list): Cá thể cha mẹ thứ hai.

    Returns:
    list: Cá thể con sau khi lai ghép.
    """
    crossover_point = len(parent1) // 2
    child = parent1[:crossover_point] + parent2[crossover_point:]
    return child
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Đột biến (mutation)

```python
def mutation(parent, mutation_rate):
    """
    Thực hiện toán tử đột biến (mutation) trên một cá thể cha mẹ.

    Parameters:
    parent (list): Cá thể cha mẹ.
    mutation_rate (float): Tỷ lệ đột biến.

    Returns:
    list: Cá thể con sau khi đột biến.
    """
    child = parent[:]  # Sao chép cá thể cha mẹ
    for i in range(len(child)):
        if random.random() < mutation_rate:
            child[i] = random.choice([0, 1])  # Thực hiện đột biến
    return child
```

4. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Sử dụng hàm crossover và hàm mutation để tạo ra quần thể mới

```
for _ in range(POP_SIZE):
  parent1, parent2 = roulette_wheel_selection(population, fitness_values)
  # Lai ghép
  child = crossover(parent1, parent2)

  # Đột biến
  mutated_child = mutation(child, MUT_RATE)

  # Thêm con vào quần thể mới
  new_population.append(mutated_child)

# Cập nhật quần thể mới
population = new_population
```

5. Hiện thực hệ thống

Genetic Algorithm for Feature Selection (GAbFS)

- Tìm kiếm chromosome tốt nhất

```python
def find_best_chromosome(data, population, num_generations, max_generations_without_improvement):
    best_fitness = -float('inf')  # Khởi tạo best_fitness với giá trị âm vô cực
    best_chromosome = None

    num_generations_without_improvement = 0

    # Vòng lặp qua các thế hệ
    for generation in range(num_generations):
        # Tính toán fitness cho mỗi cá thể trong quần thể
        fitness_values = calculate_fitness_values(data, population)

        # Tìm fitness tốt nhất trong thế hệ hiện tại
        max_fitness_in_generation = max(fitness_values)

        # Kiểm tra xem có sự cải thiện so với best_fitness hay không
        if max_fitness_in_generation > best_fitness:
            best_fitness = max_fitness_in_generation
            best_chromosome_index = fitness_values.index(max_fitness_in_generation)
            best_chromosome = population[best_chromosome_index]
            num_generations_without_improvement = 0
        else:
            num_generations_without_improvement += 1

        # Kiểm tra tiêu chí dừng: số thế hệ không có cải thiện vượt quá ngưỡng cho phép
        if num_generations_without_improvement >= max_generations_without_improvement:
            break

    return best_chromosome
```

5. Hiện thực hệ thống

Phân loại sử dụng các bộ phân loại SVM, KNN, và XGBoost

```python
def classification(data, best_chromosome, classifier, num_folds=5, threshold=0.5):
    """
    Hàm thực hiện quá trình phân loại sử dụng các bộ phân loại khác nhau.

    Parameters:
    - data (DataFrame): Dữ liệu đầu vào.
    - best_chromosome (list): Tập hợp các đặc trưng tối ưu.
    - classifier (str): Bộ phân loại được sử dụng, có thể là 'svm', 'knn', hoặc 'xgboost'.
    - num_folds (int): Số lượng fold trong cross-validation.
    - threshold (float): Ngưỡng phân loại.

    Returns:
    - accuracy (float): Độ chính xác trung bình của mô hình.
    - recall (float): Recall trung bình của mô hình.
    """
    # Lựa chọn các đặc trưng từ tập dữ liệu dựa trên best_chromosome
    selected_features = data.iloc[:, best_chromosome]
    X = selected_features.values
    y = data['Label'].values  # Giả sử cột 'Label' là cột nhãn

    # Sử dụng SMOTE để làm cân bằng dữ liệu
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)

    # Chia dữ liệu thành tập huấn luyện và tập kiểm tra
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.5, random_state=42)
```

5. Hiện thực hệ thống

Phân loại sử dụng các bộ phân loại SVM, KNN, và XGBoost

```
# Chọn bộ phân loại
if classifier == 'svm':
    clf = SVC(
            kernel='linear',    # Sử dụng kernel tuyến tính để phân loại dữ liệu.
            C=1.0,              # Hệ số phạt lỗi tiêu chuẩn, cân bằng giữa độ chính xác và khả năng tổng quát hóa.
            gamma='scale',      # Hệ số gamma, tuy nhiên, với kernel tuyến tính, tham số này không được sử dụng.
            class_weight='balanced', # Điều chỉnh trọng số lớp tự động để xử lý dữ liệu không cân bằng.
            probability=True    # Tính toán xác suất dự đoán.
        )
elif classifier == 'knn':
     clf = KNeighborsClassifier(
            n_neighbors=9,        # Số lượng láng giềng gần nhất được sử dụng để dự đoán.
            weights='distance',   # Láng giềng gần hơn có trọng số cao hơn.
            algorithm='auto',     # Tự động chọn thuật toán tối ưu nhất cho dữ liệu.
            p=2,                  # Sử dụng khoảng cách Euclidean (L2).
            metric='minkowski'    # Sử dụng khoảng cách Minkowski (tổng quát).
     )
elif classifier == 'xgboost':
    clf = XGBClassifier(
            learning_rate=0.1,    # Tốc độ học của mô hình, giúp làm chậm quá trình học để tránh overfitting.
            n_estimators=100,     # Tăng số lượng cây để cải thiện hiệu suất và độ chính xác của mô hình.
            max_depth=5,          # Độ sâu tối đa của mỗi cây, giữ nguyên để cân bằng giữa độ chính xác và khả năng tổng quát hóa.
            gamma=0,              # Giảm giá trị gamma để cho phép chia tách nhiều hơn, cải thiện khả năng học của mô hình.
            subsample=0.8,        # Tăng tỷ lệ mẫu được sử dụng để xây dựng mỗi cây, giảm khả năng underfitting.
            colsample_bytree=0.8  # Tăng tỷ lệ cột được sử dụng để xây dựng mỗi cây, giảm khả năng underfitting.
        )
```

5. Hiện thực hệ thống

Kết quả

	Classifier / Tiêu chí	k-NN	XgBoost	SVM
CIRA-CIC-DOHBrw-2020	Accuracy	0.9124	0.9463	0.9533
	Recall	0.9123	0.9463	0.9530
UNSW-NB15	Accuracy	0.9478	0.9656	0.9463
	Recall	0.9163	0.9098	0.9346
Bot-IoT	Accuracy	0.9466	0.9787	0.9854
	Recall	0.9453	0.9721	0.9832

4. Tài liệu tham khảo

- Halim, Z., Yousaf, M. N., Waqas, M., Sulaiman, M., Abbas, G., Hussain, M., ... & Hanif, M. (2021). An effective genetic algorithm-based feature selection method for intrusion detection systems. Computers & Security, 110, 102448.
- Yusof, M. H. M., Almohammedi, A. A., Shepelev, V., & Ahmed, O. (2022). Visualizing realistic benchmarked ids dataset: Cira-cic-dohbrw-2020. IEEE Access, 10, 94624-94642.
- Peterson, J. M., Leevy, J. L., & Khoshgoftaar, T. M. (2021, August). A review and analysis of the bot-iot dataset. In 2021 IEEE International Conference on Service-Oriented System Engineering (SOSE) (pp. 20-27). IEEE.
- Moustafa, N., & Slay, J. (2015, November). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In 2015 military communications and information systems conference (MilCIS) (pp. 1-6). IEEE.

Cảm ơn thầy và các bạn đã lắng nghe bài thuyết trình của nhóm.