|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” | Mẫu 2 |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.4.2

“Phương pháp sinh đặc trưng FP-Growth”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

Hà Nội - 2023

|  |  |  |
| --- | --- | --- |
|  | BAN CƠ YẾU CHÍNH PHỦ  “HỌC VIỆN KỸ THUẬT MẬT MÃ” |  |

BÁO CÁO CHUYÊN ĐỀ SỐ 4.4.2

“Phương pháp sinh đặc trưng FP-Growth”

NHIỆM VỤ: “Nghiên cứu và ứng dụng nền tảng học sâu để xây dựng hệ thống phát hiện mã độc trực tuyến”.

Mã số: 06/2022/CB.

Cơ quan chủ trì: Học viện Kỹ thuật Mật mã

Chủ nhiệm: ThS. Lê Đức Thuận

|  |  |
| --- | --- |
| **Người thực hiện chuyên đề** | **Cơ quan chủ trì** |
| *(Họ tên và chữ ký)* | *(Họ tên và chữ ký)* |

Hà Nội - 2023

MỤC LỤC

[MỤC LỤC 1](#_Toc129698658)

[DANH MỤC HÌNH ẢNH 1](#_Toc129698659)

[PHƯƠNG PHÁP SINH ĐẶC TRƯNG FP-GROWTH 2](#_Toc129698660)

[1.1. Quy trình phát hiện mã độc sử dụng FP-Growth 2](#_Toc129698661)

[1.2. Xây dựng mô hình toán học 2](#_Toc129698662)

[1.3. Thuật toán FP-Gowth 3](#_Toc129698663)

[1.4. Cải tiến tập đặc trưng dựa trên FP-Growth 5](#_Toc129698664)

[1.5. Code 7](#_Toc129698665)

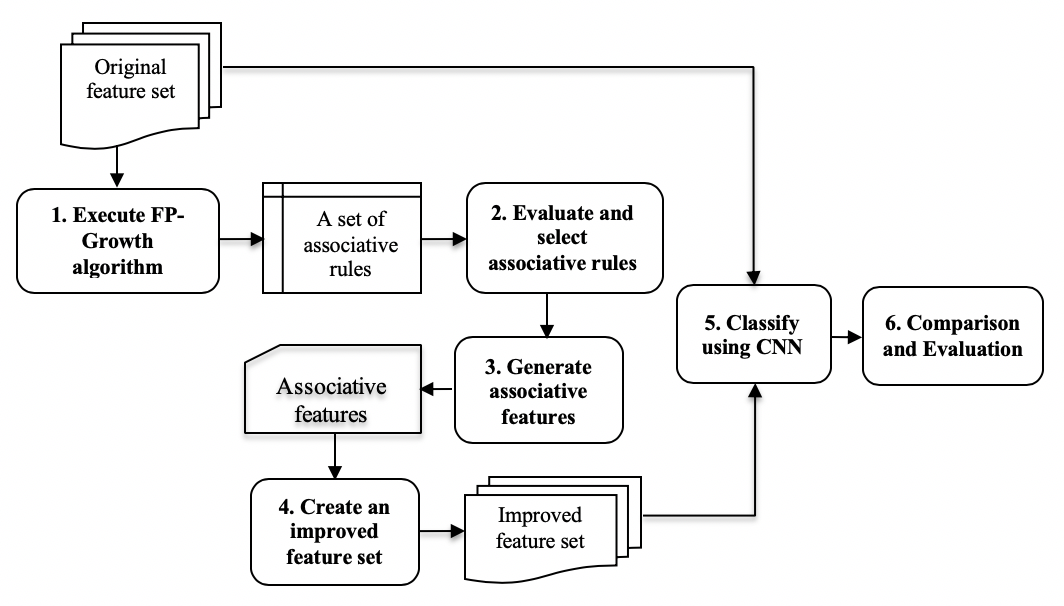
# DANH MỤC HÌNH ẢNH

[Hình 1: Phát hiện mã độc sử dụng FP-Growth 2](file:///D:\DT_Ban%202022\bản%20word\Báo%20cáo%20tiến%20độ\Nội%20dung%204\BC_CĐ%204.4.2.%20Phương%20pháp%20sinh%20đặc%20trưng%20FP-Growth.docx#_Toc129698477)

# PHƯƠNG PHÁP SINH ĐẶC TRƯNG FP-GROWTH

## Quy trình phát hiện mã độc sử dụng FP-Growth

Để phát triển phương pháp cải tiến tập đặc trưng theo ý tưởng trên, chúng tôi tiến hành nghiên cứu và thực nghiệm theo quy trình trong Hình 1. Đầu tiên, tập đặc trưng ban đầu được khai phá để tìm ra các luật kết hợp; mỗi luật kết hợp thể mối quan hệ giữa một nhóm đặc trưng. Lựa chọn các luật kết hợp phù hợp để tạo ra đặc trưng mới. Bổ sung các đặc trưng mới vào tập đặc trưng ban đầu để tạo thành tập đặc trưng cải tiến. Tập đặc trưng ban đầu và tập đặc trưng cải tiến được mã hoá để đưa vào CNN. Đánh giá kết quả phân lớp trên CNN từ tập đặc trưng ban đầu và tập đặc trưng cải tiến.



Hình 1: Phát hiện mã độc sử dụng FP-Growth

## Xây dựng mô hình toán học

Để phát triển và thử nghiệm phương pháp cải tiến đặc trưng, chúng tôi tiến hành xây dựng mô hình toán học tổng thể của bài toán. Cách tiếp cận trong phương pháp đề xuất là sinh, đánh giá các đặc trưng mới từ tập đặc trưng ban đầu dựa trên quan hệ kết hợp giữa các nhóm đặc trưng. Các đặc trưng mới này gọi là các đặc trưng kết hợp sẽ được bổ sung vào tập đặc trưng ban đầu để tạo thành tập đặc trưng cải tiến. Luật kết hợp đặc trưng và đặc trưng kết hợp được định nghĩa như sau:

**Định nghĩa 1 – Luật kết hợp đặc trưng**

Luật kết hợp đặc trưng là mối quan hệ cùng tồn tại giữa hai tập đặc trưng con trong tập mẫu.

**Định nghĩa 2 – Đặc trưng kết hợp**

Đặc trưng kết hợp là đặc trưng mới được tạo ra từ hai tập đặc trưng con và luật kết hợp giữa chúng với một ngưỡng tin cậy cụ thể.

Mô hình toán học tổng thể của bài toán cải tiến tập đặc trưng trong phát hiện tấn công IoT được mô tả như trong công thức (1).

|  |  |
| --- | --- |
| *FA = F U FN*  *FN = {a | a = fG(F1, F2, r)}* | (1) |

Trong đó,

* *F* là tập đặc trưng ban đầu
* *FA* là tập đặc trưng cải tiến
* *FN* là tập các đặc trưng mới
* *a* là một đặc trưng kết hợp
* *F1, F2* là các tập đặc trưng con
* *r* là một luật kết hợp đặc trưng
* *fG* là hàm sinh đặc trưng kết hợp

Căn cứ vào mô hình tổng thể của bài toán, các bước xây dựng, phân tích, thử nghiệm và đánh giá phương pháp sẽ được trình bày cụ thể trong các phần tiếp sau.

## Thuật toán FP-Gowth

Thuật toán FP-Growth xây dựng cấu trúc FP tree để lưu trữ toàn bộ cơ sở dữ liệu và khai phá FP tree để tìm các mẫu phổ biến. Việc nén tập dữ liệu vào cấu trúc cây (Frequent Pattern tree – FP-tree) làm giảm chi phí cho toàn tập dữ liệu trong quá trình khai phá, các infrequent items bị loại bỏ sớm mà vẫn đảm bảo kết quả khai phá không bị ảnh hưởng và tránh được việc tạo ra các tập dự tuyển trong mỗi lần kiểm tra dữ liệu như với thuật toán Apriori. Với những đặc điểm đó, thuật toán FP-Growth có tốc độ xử lý nhanh hơn nhiều so với thuật toán Apriori.

**Cấu trúc FP tree:**

* Nút gốc giá trị Null (cấp độ 0)
* Thông tin mỗi nút: <name, support, parent\_node>
* Trên cùng một nhánh, nút cấp độ k có giá trị support lớn hơn hoặc bằng nút cấp độ k+1 (1<=k<=N, N là số lượng item phổ biến)

**Giải thuật xây dựng FP tree:**

Input:

D, a transaction database

Min\_sup, the minimum support count threshold

Output: The complete set of frequent patterns

Thuận toán:

// 1. The FP-tree is constructed in the following steps:

***for each*** *transaction in D {*

*Tìm item\_i đưa vào tập item phổ biến F;*

*Tính support của item\_i }*

//Sắp xếp F theo thứ tự giảm dần support tạo thành tập phổ biến L

*L=F.sort(item, support);*

//Build T tree

*Initialize T; root\_node = Null; root\_node.name=Null;*

***For each*** *transaction in D {*

*Select item in transaction; L.sort(item, support);*

*Insert\_tree([p|P],T) }*

//Thủ tục *Insert\_tree*([p|P],T) như sau:

***If*** *(T có node con N & N.name=item.name)*

*N.count++*

***Else*** *{*

*Tạo node N mới; N.parent=T; N.count=1 }*

***If*** *(P<>rỗng)*

*Insert\_tree(P, N)*

//2. Khám phá frequent itemsets với FP-tree được thực hiện bằng cách gọi thủ tục *FP-Growth(FP\_tree, null)*

***Procedure FP-Growth(Tree,α)***

***If Tree*** *contains a single path* ***P then***

***For each*** *combination (denoted as β) of the nodes in P*

*{Generate pattern 𝛽; support = minimum support of nodes in 𝛽 }*

***Else for each*** *in the header of* ***Tree*** *{*

*Generate pattern ; support = .support;*

*Construct 𝛽’s conditional pattern base;*

*Construct 𝛽’s conditional FP\_tree*

***If***  *≠ø* ***then***

*Call* ***FP-Growth(, β)*** *}*

## Cải tiến tập đặc trưng dựa trên FP-Growth

**Khai phá, đánh giá, lựa chọn tập luật kết hợp đặc trưng**

Dựa trên thuật toán FP-Growth trong phần trước, chúng tôi thực hiện việc khai phá dữ liệu trên tập đặc trưng ban đầu để tìm ra các luật kết hợp. Theo định nghĩa 1 và công thức (1), tập luật kết hợp đặc trưng R được mô tả như trong công thức (2).

*R =* {*r | r: F1 => F2; F1, F2 ⊂ F*} (1)

Để thực hiện thuật toán FP-Growth, cần phải chuyển các đặc trưng ban đầu theo định dạng đầu vào của thuật toán. Sau khi thực thi thuật toán, thu được tập luật kết hợp đặc trưng trong đó mỗi luật mô tả mối quan hệ giữa hai nhóm đặc trưng. Tuy nhiên, tập luật này có kích thước lớn và chứa nhiều luật yếu nên cần đánh giá để lựa chọn các luật mạnh. Việc lựa chọn các luật mạnh dựa trên độ tin cậy và độ hỗ trợ. Dựa trên các độ đo này, chúng tôi được tập luật kết hợp đặc trưng R’ là tập con của R, thoả mãn công thức (3).

*R’ =* {*r’ | r’ ∈ R; r’.conf >= min\_conf; r’.sup >= min\_sup*} (2)

**Sinh các đặc trưng kết hợp**

Các đặc trưng kết hợp, thể hiện mối quan hệ giữa các tập đặc tưng, sẽ được sinh ra từ tập luật kết hợp đặc trưng. Theo định nghĩa 2 và mô tả trong công thức (1), mỗi đặc trưng kết hợp được sinh ra dựa trên hàm sinh fG(F1, F2, r). Hàm fG này được xây dựng như trong công thức (4):

|  |  |
| --- | --- |
|  | (3) |

Trong đó,

* *ai* là đặc trưng thứ *i* trong tập con đặc trưng *F1*
* *bi* là đặc trưng thứ *i* trong tập con đặc trưng F2
* *rconf* là độ tin cậy của luật *r*
* *rsup* là độ hỗ trợ của luật *r*

**Tạo tập đặc trưng cải tiến**

Các đặc trưng kết hợp được bổ sung vào tập đặc trưng ban đầu để tạo tập đặc trưng cải tiến. Với tập đặc trưng ban đầu, các đặc trưng độc lập với nhau do đó không thể hiện được sự tương quan giữa các đặc trưng hoặc các nhóm đặc trưng. Với tập đặc trưng cải tiến, mối quan hệ giữa các nhóm đặc trưng đã được thể hiện trong các đặc trưng kết hợp. Phương pháp đánh giá mức độ cải tiến của tập đặc được trình bày trong phần sau.

## Code

from collections import defaultdict, namedtuple

import time

def find\_frequent\_itemsets(transactions, minimum\_support, include\_support=False):

items = defaultdict(lambda: 0)

for transaction in transactions:

for item in transaction:

items[item] += 1

items = dict((item, support) for item, support in items.items()

if support >= minimum\_support)

def clean\_transaction(transaction):

transaction = filter(lambda v: v in items, transaction)

transaction\_list = list(transaction) # 为了防止变量在其他部分调用，这里引入临时变量transaction\_list

transaction\_list.sort(key=lambda v: items[v], reverse=True)

return transaction\_list

#FPtree

master = FPTree()

for transaction in map(clean\_transaction, transactions):

master.add(transaction)

def find\_with\_suffix(tree, suffix):

for item, nodes in tree.items():

support = sum(n.count for n in nodes)

if support >= minimum\_support and item not in suffix:

found\_set = [item] + suffix

yield (found\_set, support) if include\_support else found\_set

cond\_tree = conditional\_tree\_from\_paths(tree.prefix\_paths(item))

for s in find\_with\_suffix(cond\_tree, found\_set):

yield s

for itemset in find\_with\_suffix(master, []):

yield itemset

class FPTree(object):

Route = namedtuple('Route', 'head tail')

def \_\_init\_\_(self):

self.\_root = FPNode(self, None, None)

self.\_routes = {}

@property

def root(self):

return self.\_root

def add(self, transaction):

point = self.\_root

for item in transaction:

next\_point = point.search(item)

if next\_point:

next\_point.increment()

else:

next\_point = FPNode(self, item)

point.add(next\_point)

self.\_update\_route(next\_point)

point = next\_point

def \_update\_route(self, point):

assert self is point.tree

try:

route = self.\_routes[point.item]

route[1].neighbor = point # route[1] is the tail

self.\_routes[point.item] = self.Route(route[0], point)

except KeyError:

# 开启新节点

self.\_routes[point.item] = self.Route(point, point)

def items(self):

for item in self.\_routes.keys():

yield (item, self.nodes(item))

def nodes(self, item):

try:

node = self.\_routes[item][0]

except KeyError:

return

while node:

yield node

node = node.neighbor

def prefix\_paths(self, item):

def collect\_path(node):

path = []

while node and not node.root:

path.append(node)

node = node.parent

path.reverse()

return path

return (collect\_path(node) for node in self.nodes(item))

def inspect(self):

#print('Tree:')

self.root.inspect(1)

#print

#print('Routes:')

for item, nodes in self.items():

#print(' %r' % item)

for node in nodes:

print(' %r' % node)

def conditional\_tree\_from\_paths(paths):

tree = FPTree()

condition\_item = None

items = set()

for path in paths:

if condition\_item is None:

condition\_item = path[-1].item

point = tree.root

for node in path:

next\_point = point.search(node.item)

if not next\_point:

# Add a new node to the tree.

items.add(node.item)

count = node.count if node.item == condition\_item else 0

next\_point = FPNode(tree, node.item, count)

point.add(next\_point)

tree.\_update\_route(next\_point)

point = next\_point

assert condition\_item is not None

for path in tree.prefix\_paths(condition\_item):

count = path[-1].count

for node in reversed(path[:-1]):

node.\_count += count

return tree

class FPNode(object):

def \_\_init\_\_(self, tree, item, count=1):

self.\_tree = tree

self.\_item = item

self.\_count = count

self.\_parent = None

self.\_children = {}

self.\_neighbor = None

def add(self, child):

if not isinstance(child, FPNode):

raise TypeError("Can only add other FPNodes as children")

if not child.item in self.\_children:

self.\_children[child.item] = child

child.parent = self

def search(self, item):

try:

return self.\_children[item]

except KeyError:

return None

def \_\_contains\_\_(self, item):

return item in self.\_children

@property

def tree(self):

return self.\_tree

@property

def item(self):

return self.\_item

@property

def count(self):

return self.\_count

def increment(self):

if self.\_count is None:

raise ValueError("Root nodes have no associated count.")

self.\_count += 1

@property

def root(self):

return self.\_item is None and self.\_count is None

@property

def leaf(self):

return len(self.\_children) == 0

@property

def parent(self):

return self.\_parent

@parent.setter

def parent(self, value):

if value is not None and not isinstance(value, FPNode):

raise TypeError("A node must have an FPNode as a parent.")

if value and value.tree is not self.tree:

raise ValueError("Cannot have a parent from another tree.")

self.\_parent = value

@property

def neighbor(self):

return self.\_neighbor

@neighbor.setter

def neighbor(self, value):

if value is not None and not isinstance(value, FPNode):

raise TypeError("A node must have an FPNode as a neighbor.")

if value and value.tree is not self.tree:

raise ValueError("Cannot have a neighbor from another tree.")

self.\_neighbor = value

@property

def children(self):

return tuple(self.\_children.itervalues())

def inspect(self, depth=0):

#print((' ' \* depth) + repr(self))

for child in self.children:

child.inspect(depth + 1)

def \_\_repr\_\_(self):

if self.root:

return "<%s (root)>" % type(self).\_\_name\_\_

return "<%s %r (%r)>" % (type(self).\_\_name\_\_, self.item, self.count)

dataset = []

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

frequent\_itemsets = find\_frequent\_itemsets(dataset, minimum\_support=1, include\_support=True)

#print(type(frequent\_itemsets)) # print type

result = []

for itemset, support in frequent\_itemsets:

result.append((itemset, support))

result = sorted(result, key=lambda i: i[0])

for itemset, support in result:

print(str(itemset) + ' ' + str(support))

end = time.time()

print('time：', str(end - start))