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

BÁO CÁO CHUYÊN ĐỀ SỐ 4.4.1

“Phương pháp sinh đặc trưng Apriori”

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 4.4.1

“Phương pháp sinh đặc trưng Apriori”

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](#_Toc129698356)

[TRÍCH XUẤT ĐẶC TRƯNG BẰNG THUẬT TOÁN APRIORI KHAI PHÁ LUẬN KẾT HỢP 2](#_Toc129698357)

[1.1. Phát biểu bài toán 2](#_Toc129698358)

[1.2. Thuật toán Apriori 2](#_Toc129698359)

[1.2.1. Các khái niệm liên quan 2](#_Toc129698360)

[1.2.2. Tư tưởng thuật toán Apriori 3](#_Toc129698361)

[2.2.3. Thuật toán Apriori 3](#_Toc129698362)

[1.3. Áp dụng Apriori trong trích chọn đặc trưng trên Android 5](#_Toc129698363)

[1.4. Kết quả 7](#_Toc129698364)

[1.5. Code 8](#_Toc129698365)

# TRÍCH XUẤT ĐẶC TRƯNG BẰNG THUẬT TOÁN APRIORI KHAI PHÁ LUẬN KẾT HỢP

## 1.1. Phát biểu bài toán

Apriori là thuật toán được Rakesh Agrawal, Tomasz Imielinski, Arun Swami đề xuất lần đầu vào năm 1993. Bài toán được phát biểu: Tìm t có độ hỗ trợ s thỏa mãn s  s0 và độ tin cậy c  c0 (s0, c0 là hai ngưỡng do người dùng xác định và s0=minsupp, c0 =minconf­) . Ký hiệu Lk tập các tập k - mục phổ biến, Ck tập các tập k-mục ứng viên.

Đặt bài toán :

1. *Tìm tất cả các tập mục phổ biến với minsupp nào đó.*
2. *Sử dụng các tập mục phổ biến để sinh ra các luật kết hợp với độ tin cậy minconf nào đó.*

## 1.2. Thuật toán Apriori

### 1.2.1. Các khái niệm liên quan

* ***Khái niệm 1* – Luật kết hợp đặc trưng**

Luật kết hợp để chỉ ra các mối quan hệ tương quan giữa 2 nhóm đặc trưng trong tập đặc trưng ban đầu. Mỗi nhóm đặc trưng là một tập con khác rỗng các đặc trưng. Với hai tập đặc trưng con X và Y, luật kết hợp đặc trưng được mô tả như trong công thức (1). Tính đúng đắn của luật kết hợp đặc trưng dựa trên độ hỗ trợ và độ tin cậy. Các độ đo này được tính như trong công thức (2), (3).

(1)

|  |  |
| --- | --- |
|  | (2) |
|  | (3) |

ở đây,

* N là tổng số giao dịch
*  là số giao dịch có 
* X.count là số giao dịch chứa X
* ***Khái niệm 2* – Đặc trưng kết hợp**

Đặc trưng kết hợp là đặc trưng được tạo ra dựa trên luật kết hợp đặc trưng và thỏa mãn ngưỡng xác định của độ tin cậy và độ hỗ trợ. Dựa trên luật kết hợp đặc trưng trong công thức (1), chúng tôi xây dựng công thức tính đặc trưng kết hợp *fm* như trong công thức (4).

(4)

* ***Khái niệm 3* – Tập đặc trưng kết hợp**

Tập đặc trưng cải tiến, ký hiệu là FC, là hợp của tập đặc trưng ban đầu và các đặc trưng kết hợp. FC được xây dựng theo công thức (5).

(5)

ở đây,

* *FA* là tập đặc trưng ban đầu
* *FM* là tập đặc trưng kết hợp

Xây dựng tập đặc trưng cải tiến sử dụng thuật toán Apriori

Các giá trị *minsup* và *mincof* là ngưỡng cần phải xác định trước khi sinh ra các luật kết hợp. Một itemsets mà *tần suất xuát hiện của nó  minsup* gọi là ***frequent itemsets.***

### 1.2.2. Tư tưởng thuật toán Apriori

* Tìm tất cả *frequent itemsets*: sử dụng *k-itemset* (itemsets gồm *k* items) được dùng để tìm *(k+1) itemset*.
* Tìm tất cả các luật kết hợp từ các *frequent itemsets* (thỏa mãn 2 tham số *minsup* và *mincof*).

### 2.2.3. Thuật toán Apriori

Thuật toán được chia thành 2 giai đoạn

- **Giai đoạn 1**: Đầu tiên tìm *1-itemset* (ký hiệu F1). F1 được dùng để tìm F2 (2-itemsets). F2 được dùng để tìm F3 (3-itemset) và tiếp tục cho đến khi không có k-itemset được tìm thấy

1 **Function** candidate-gen (Fk-1)

2  //initialize the set of candidetes

3 **forall** f1, f2 Fk-1 //find all pairs of frequent itemsets

4 with f1 = {i1, …, ik-2, ik-1} //that differ only in the

5 and f2 = {i1, …, ik-2, i’k-1} //last item

6 and ik-1 < i’k-1 **do**

7//according to the lexicographic order

8 {i1, …, ik-2, i’k-1};

9 //join the two itemsets f1 and f2.

10 

11 //add the new itemset *c* to the candidates

12 **for** each (k-1)-subset s of c **do**

13 **If** () **then**

14 delete *c* from *Ck*;

15 //delete *c* from the candidates

16 **endfor**

17 **endfor**

18 **return** Ck; //return the generated candidates

- **Giai đoạn 2**: Sử dụng các *frequent itemsets* thu được ở bước 1 sinh ra các luật kết hợp thỏa mãn *confidence ≥ minconf*.

1 **Algorithm** genRules (*F*)

2 //*F* is the set of all frequent itemsets

2 **for** each frequent k-itemset *fk* in *F, k*  2 **do**

3 output every 1-item consequent rule of *fk* with

4 confidence  *minconf* and *support fk.count /n*

5 //*n* is the total number of transactions in *T*

6 *H1* {consequents of all 1-item consequent rules

7 derived from *fk* above};

8 ap-genRules(fk, H1);

9 **endfor**

10 **Procedure** ap-genRules(*fk, Hm*)

11 //*Hm* is the set of *m*-item consequents

12 ***if*** (*k > m+1*) AND (*)* **then**

13 *Hm+1*  candidate-gen (*Hm*);

14 **for each** *hm+1* in *Hm+1* **do**

15 *conf  fk.count / (fk – hm+1).count;*

16 **if** (*conf minconf*) **then**

17 output the rule *(fk – hm+1*) *hm+1* with

18 confidence *= conf*  and support = *fk.count /n*;

19 //*n* is the total number of trancactions in *T*

20 **else**

21 delete *hm+1* from *Hm+1*;

22 endfor

23 ap-genRules (*fk, Hm+1*);

24 **endif**

## 1.3. Áp dụng Apriori trong trích chọn đặc trưng trên Android

Sau khi thực hiện trích xuất ở chương 1, có các tập đặc trưng của mỗi mẫu gồm các đặc trưng là Permission, Api, String, Intent, Activities, ... Tuy nhiên trong chuyên đề này chúng tôi sử dụng Permission, API là dữ liệu sử dụng trong thuật toán Apriori do các đặc trưng khác có tính chuyên biệt hóa cao, một đặc trưng ở tập tin này ít xuất hiện ở tập tin khác.

Tập dữ liệu sử dụng là:

* Drebin với 5560 file mã độc gồm 179 họ được gán mã từ 1-179
* Tập mã sạch gồm 7140 file được gán nhãn 0

Với đặc trưng Permission được giới hạn với nhà phát hành hệ điều hành gồm 324 quyền cộng với các quyền được tự định nghĩa, tổng cộng các quyền được dùng là 398 xuất hiện trên tất cả các tập tin.

Với đặc trưng API không có giới hạn nào nên sẽ lấy rank 200 API có tần suất nhiều nhất trong tập dữ liệu .apk

* **Mô tả các tập dữ liệu**

Bộ dữ liệu được lưu dưới dạng đuôi mở rộng .csv trong đó phần dữ liệu được mô tả dưới dạng nhị phân 0 hoặc 1.

Minsup được lấy giá trị là 0.45 để chạy trong trích chọn đặc trưng và số vòng thực hiện apriori là 3

Mỗi dòng dữ liệu là mỗi mẫu được sử dụng trong tập dữ liệu

Cột 0 là nhãn của các nhãn được đánh số từ 0-179

Cột 1…n là biểu thị cho đặc trưng, như vậy sẽ có *n* đặc trưng. Mẫu nào có đặc trưng đó thì được đánh là 1 nếu không có thì ngược lại là 0

* **Bộ 1 gồm Permission + API**

Trong bộ dữ liệu này, sử dụng thuần túy hai nhóm đặc trưng được sử dụng phổ biến để phát hiện mã độc là Permission và API được trích xuất từ bộ dữ liệu trên. Permission bao gồm 398 đặc trưng và API bao gồm 200 đặc trưng.

* **Bộ 2 gồm Permission + Api và Apriori của permission, Apriori của Api**

Trong bộ dữ liệu này, kết hợp đặc trưng của từng nhóm sử dụng thuật toán khai phá dữ liệu Apriori để được thêm các đặc trưng mới, là sự liên kết giữa các đặc trưng trong nhóm.

Như vậy ngoài 398 đặc trưng permission ban đầu sẽ có thêm 2132 đặc trưng kết hợp. Với 200 đặc trưng API ban đầu có thêm 3085 đặc trưng kết hợp.

Như vậy, ngoài các đặc trưng vốn có, khi sử dụng Apriori, đã tạo thêm các đặc trưng liên kết để tạo thành bộ đặc trưng mới với nhiều thông tin hơn.

## 1.4. Kết quả

Khi sử dụng hai bộ dữ liệu trên cùng mô hình học máy, cụ thể được dùng cho mô hình mạng nơ-ron tích chập. Mạng nơ-ron này được cấu hình như sau:

|  |
| --- |
| **Input**: nm1  **Layer1**: CONV1: 33 size, 32 filter, ReLU  Max Pool: 22 size  **Layer2:** CONV2: 33 size, 32 filter, ReLU  Max Pool: 22 size  **Layer3:** CONV3: 33 size, 64 filter, ReLU  Max Pool: 22 size  **FC:** 1024 Hidden Neuron  **Dropout**: 0.8  **FC:** 179 Output Classes |

Khi đó, kết quả được chạy theo 10-fold như sau:

Bảng 1:Kết quả chạy thực nghiệm

|  |  |  |
| --- | --- | --- |
| **Tập dữ liệu** | **Tỉ lệ phân lớp đúng (%)** | |
| **Bộ 1** | **Bộ 2** |
| 1 | 92.2 | 93.4 |
| 2 | 95.8 | 95.5 |
| 3 | 97.3 | 97 |
| 4 | 96.5 | 96.7 |
| 5 | 97.1 | 97.6 |
| 6 | 97.2 | 97 |
| 7 | 97.5 | 97.7 |
| 8 | 97.7 | 97.6 |
| 9 | 97.3 | 97.8 |
| 10 | 96.3 | 96.8 |
| **Trung bình** | **96.49** | **96.71** |

Như vậy, nhìn vào kết quả trung bình sau khi chạy 10 lần trên hai bộ dữ liệu (bộ 1 bao gồm các đặc trưng permission và API, Bộ hai là các đặc trưng trong bộ một và thêm các đặc trưng kết hợp khi sử dụng thuật toán Apriori), ta thấy kết quả phân lớp bộ hai cao hơn bộ 1 khi sử dụng cùng mạng nơ-ron tích chập.

## 1.5. Code

import time

def load\_data\_set():

data\_set = []

return data\_set

def create\_C1(data\_set):

C1 = set()

for t in data\_set:

for item in t:

item\_set = frozenset([item])

C1.add(item\_set)

return C1

def is\_apriori(Ck\_item, Lksub1):

for item in Ck\_item:

sub\_Ck = Ck\_item - frozenset([item])

if sub\_Ck not in Lksub1:

return False

return True

def create\_Ck(Lksub1, k):

Ck = set()

len\_Lksub1 = len(Lksub1)

list\_Lksub1 = list(Lksub1)

for i in range(len\_Lksub1):

for j in range(1, len\_Lksub1):

l1 = list(list\_Lksub1[i])

l2 = list(list\_Lksub1[j])

l1.sort()

l2.sort()

if l1[0:k-2] == l2[0:k-2]:

Ck\_item = list\_Lksub1[i] | list\_Lksub1[j]

# pruning

if is\_apriori(Ck\_item, Lksub1):

Ck.add(Ck\_item)

return Ck

def generate\_Lk\_by\_Ck(data\_set, Ck, min\_support, support\_data):

Lk = set()

item\_count = {}

for t in data\_set:

for item in Ck:

if item.issubset(t):

if item not in item\_count:

item\_count[item] = 1

else:

item\_count[item] += 1

t\_num = float(len(data\_set))

for item in item\_count:

if (item\_count[item] / t\_num) >= min\_support:

Lk.add(item)

support\_data[item] = item\_count[item] / t\_num

return Lk

def generate\_L(data\_set, k, min\_support):

support\_data = {}

C1 = create\_C1(data\_set)

L1 = generate\_Lk\_by\_Ck(data\_set, C1, min\_support, support\_data)

Lksub1 = L1.copy()

L = []

L.append(Lksub1)

for i in range(2, k+1):

Ci = create\_Ck(Lksub1, i)

Li = generate\_Lk\_by\_Ck(data\_set, Ci, min\_support, support\_data)

Lksub1 = Li.copy()

L.append(Lksub1)

return L, support\_data

def generate\_big\_rules(L, support\_data, min\_conf):

big\_rule\_list = []

sub\_set\_list = []

for i in range(0, len(L)):

for freq\_set in L[i]:

for sub\_set in sub\_set\_list:

if sub\_set.issubset(freq\_set):

conf = support\_data[freq\_set] / support\_data[freq\_set - sub\_set]

big\_rule = (freq\_set - sub\_set, sub\_set, conf)

if conf >= min\_conf and big\_rule not in big\_rule\_list:

# print freq\_set-sub\_set, " => ", sub\_set, "conf: ", conf

big\_rule\_list.append(big\_rule)

sub\_set\_list.append(freq\_set)

return big\_rule\_list

if \_\_name\_\_ == "\_\_main\_\_":

start = time.time()

data\_set = load\_data\_set()

L, support\_data = generate\_L(data\_set, k=3, min\_support=0.2)

big\_rules\_list = generate\_big\_rules(L, support\_data, min\_conf=0.7)

for Lk in L:

print ("="\*50)

print ("frequent " + str(len(list(Lk)[0])) + "-itemsets（）\t\tsupport（）")

print ("="\*50)

for freq\_set in Lk:

print (freq\_set, support\_data[freq\_set])

#print

print ("value")

for item in big\_rules\_list:

print (item[0], "=>", item[1], "conf(abc): ", item[2])

end=time.time()

print('Time：',str(end-start))