1. TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH PYTHON
2. **Tồng quan về Python và PyCharm**
3. Lịch sử phát triển về Python

* Python được Guido van Rossum tạo ra năm 1990. Python là một ngôn ngữ lập trình năng động với nhiều tính năng được sử dụng trong một loạt các ứng dụng. Python được phát triển trong một dự án Open Source do tổ chức phi lợi nhuận Python Software Foundation (PFS) quản lý.

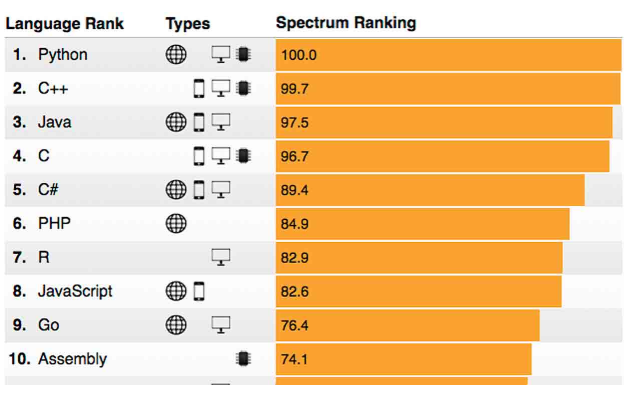


**Hình I.1: Logo ngôn ngữ Python**

Cho đến nay lịch sử phát triển Python gồm 3 giai đoạn:

* Giai đoạn 1: bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido làm việc tại CWI (*Centrum voor Wiskunde en Informatica* - Trung tâm Toán-Tin học tại Amsterdam, Hà Lan). Vì vậy, các phiên bản Python đầu tiên đều do CWI phát hành. Phiên bản cuối cùng phát hành tại CWI là 1.2.
  + Vào năm 1995, Guido chuyển sang CRNI (*Corporation for National Research Initiatives*) ở Reston, Virginia. Tại đây, ông phát hành một số phiên bản khác. Python 1.6 là phiên bản cuối cùng phát hành tại CNRI.
  + Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn GPL (*GNU General Public License –* giấy phép phần mềm tự do). Sau đó, CNRI và FSF (*Free Software Foundation* - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Cùng năm đó, Guido được nhận Giải thưởng FSF vì Sự phát triển Phần mềm tự do (*Award for the Advancement of Free Software*).
  + Phiên bản 1.6.1 ra đời sau đó là phiên bản đầu tiên tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.
* Giai đoạn 2: vào năm 2000, Guido và nhóm phát triển Python dời đến BeOpen.com và thành lập BeOpen PythonLabs team. Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập Digital Creations.
  + Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của PSF, một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.
* Giai đoạn 3: còn gọi là **Python 3000** hoặc **Py3K**: Dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python".

Hiện nay Python chạy trên rất nhiều hệ điều hành như: Windows, Linux/Unix, OS/2, Mac OS, MS-DOS, Amiga, máy ảo Java, …



**Hình I.2: Top 10 ngôn ngữ lạp trình phổ biến nhất năm 2018 [1]**

1. **Lịch sừ phát triển về PyCharm**

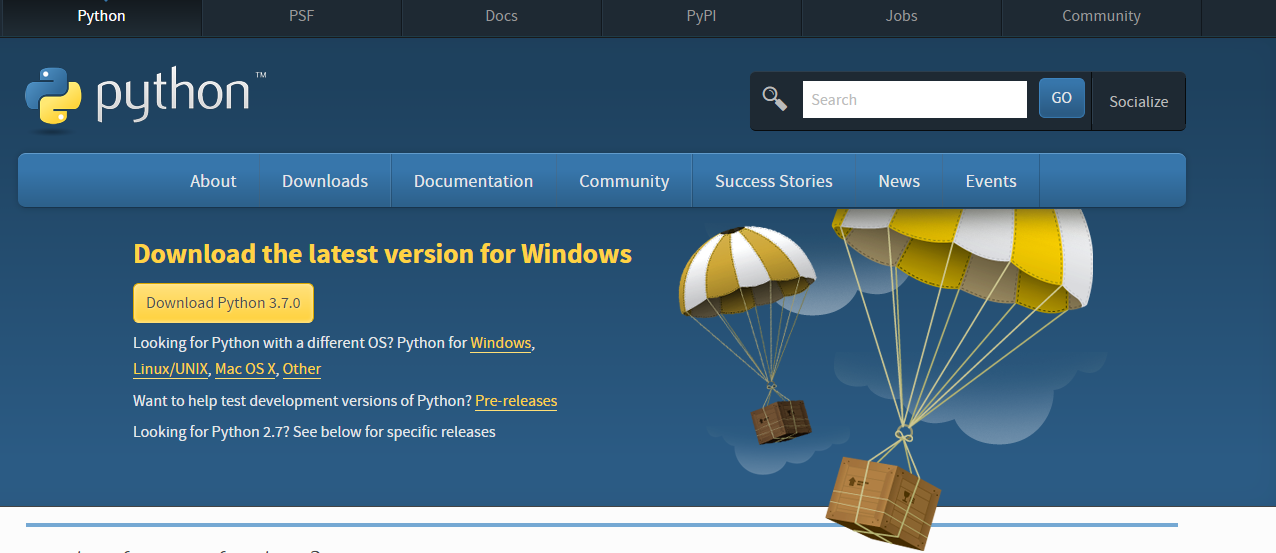
* PyCharm là một môi trường phát triển tích hợp (IDE) được sử dụng trong lập trình máy tính, đặc biệt cho ngôn ngữ Python. Nó được phát triển bởi công ty JetBrains của Séc. Nó cung cấp các chức năng như: phân tích mã, trình gỡ lỗi đồ họa, trình kiểm tra tích hợp, tích hợp với các hệ thống điều khiển phiên bản (VCSes- version control systems), và hỗ trợ phát triển web với Django.
* Phiên bản beta của PyCharm đã được phát hành vào tháng 7 năm 2010 với version 1.0, đến 3 tháng sau đó phiên bản 2.0 đã được phát hành vào ngày 13 tháng 12 năm 2011, phiên bản 3.0 vào ngày 24 tháng 9 năm 2013 và phiên bản 4.0 vào ngày 19 tháng 11 năm 2014.
* Cho đến nay PyCharm có 2 phiên bản là: Professional là phiên bản đủ (có phí), và Community (miễn phí)

1. **Cài đặt và cấu hình**

* **Yêu cầu hệ thống:**
* Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
* 2 GB RAM minimum
* 4 GB RAM recommended
* 1024x768 minimum screen resolution
* Python 2.4 or higher, Jython, PyPy or IronPython

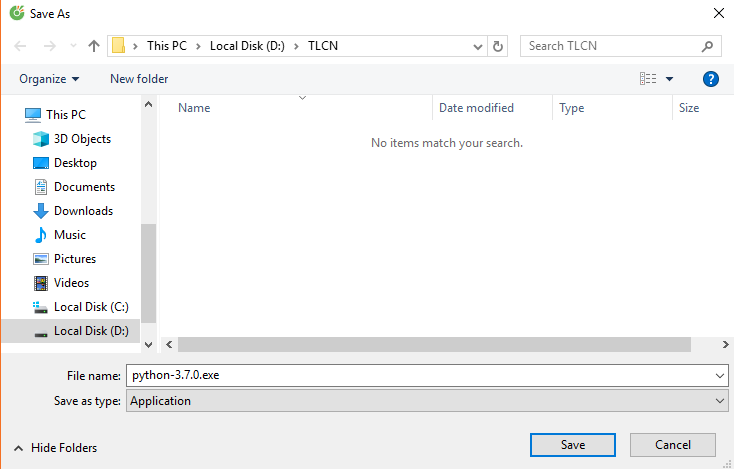
1. Cài đặt Python trên Windows

Chúng ta vào: <https://www.python.org/downloads/> để tải Python

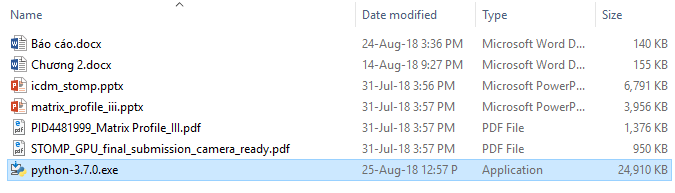


**Hình I.3: Màn hình tải Python cho Windows**

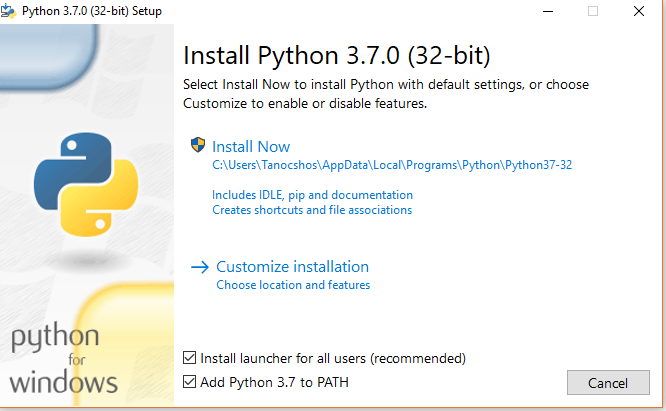
Sau khi click vào button **Download Python 3.7.0** thì ta được sẽ hiện ra thư mục để chúng ta lưu tệp **python-3.7.0.exe**



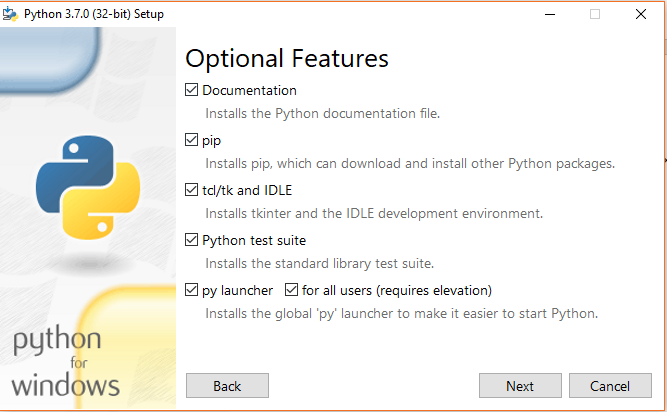
Sau khi tải xong thì chúng ta sẽ có tệp **python-3.7.0.exe** có dung lượng hơn 24MB



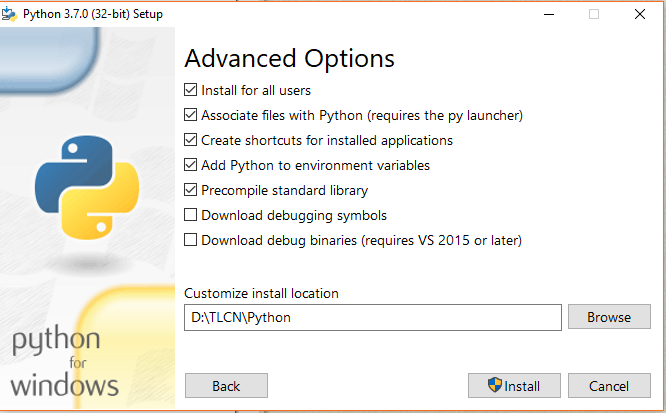
Chúng ta click vào tệp **python-3.7.0.exe** dưới đây là giao diện



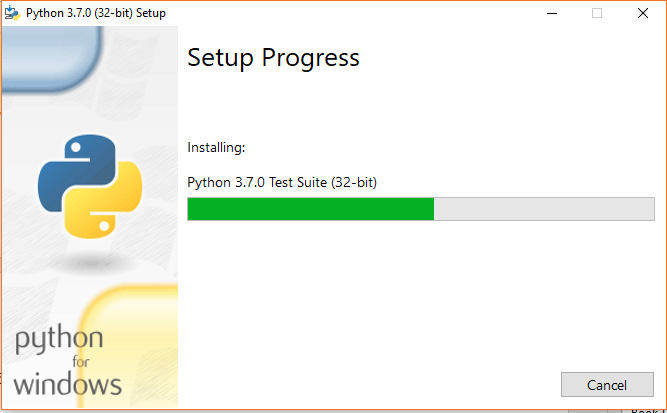
Ở trên có hai offer để chúng ta chọn 1 là click vào **Install Now** 2 là click vào **Customize installation,** ở đây chúng ta chọn **Customize installation** để thay đổi đường dẫn khác cho đơn giản và dễ nhớ.



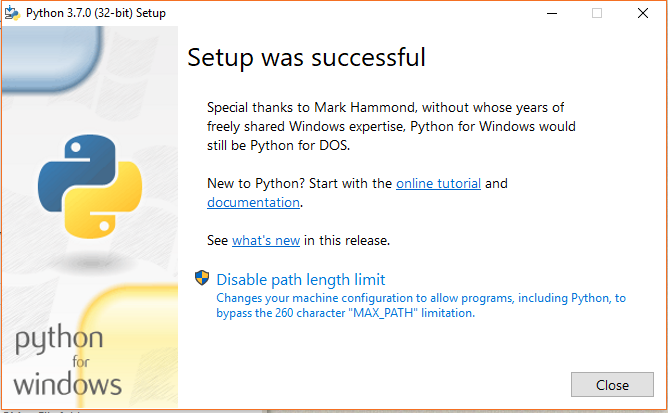
Chúng ta cấu hình như hình trên rồi click **Next**



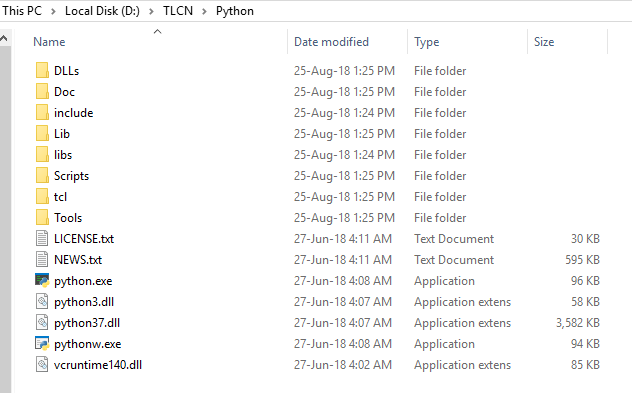
Tiếp theo lưu đường dẫn mà chúng ta muốn lưu và cấu hình như hình trên rồi click **Install**



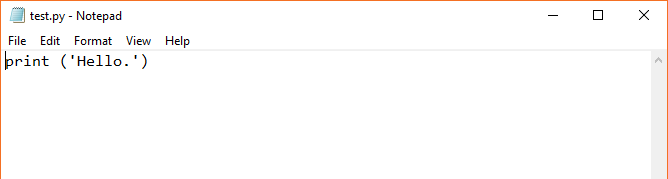
Chúng ta chờ nó chạy xong hiện cửa sổ như hình dưới đây là chúng ta hoàn thành bước cài đặt Python trên Windows



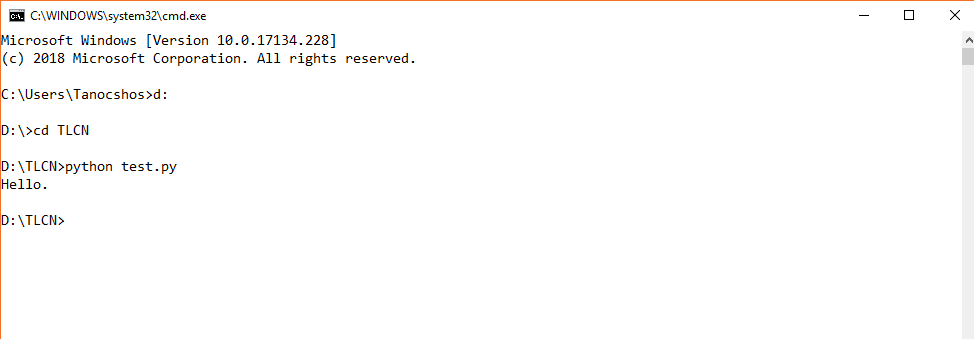
Đây là thư mục chúng ta cài đặt



Chúng ta sẽ kiểm tra xem việc cài đặt Python đã chạy được chưa bằng việc tạo 1 file test.py (dùng **Notepad** hay **Text Document** cũng được) với nội dung **print (‘Hello.’)**

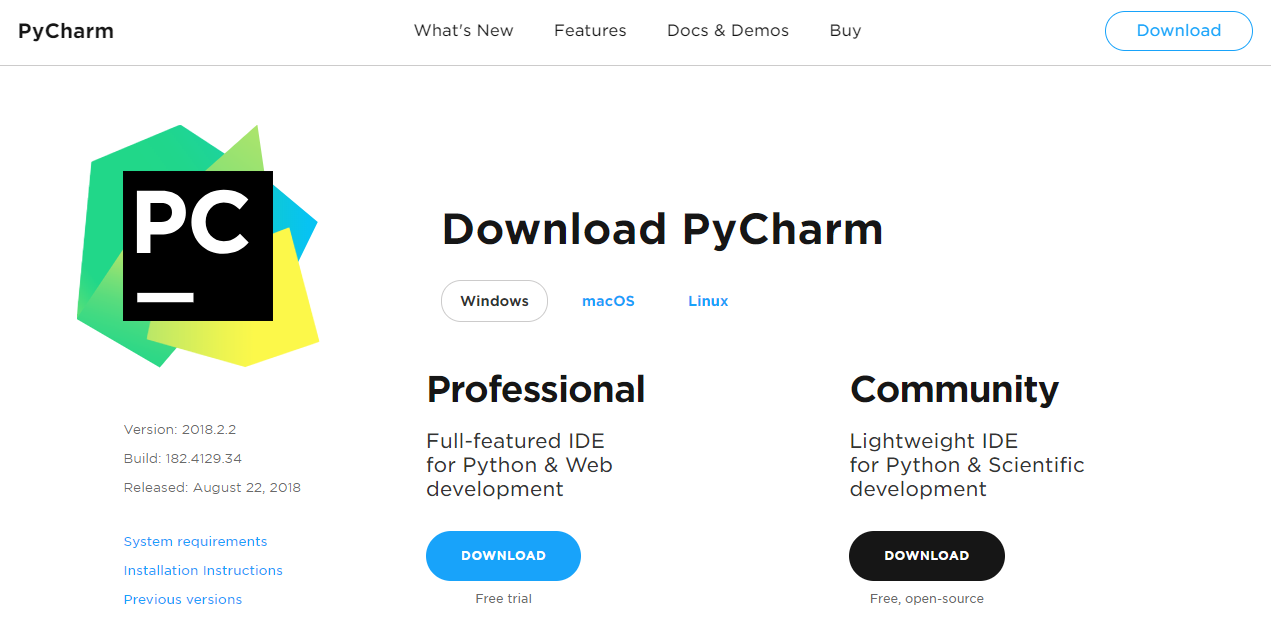


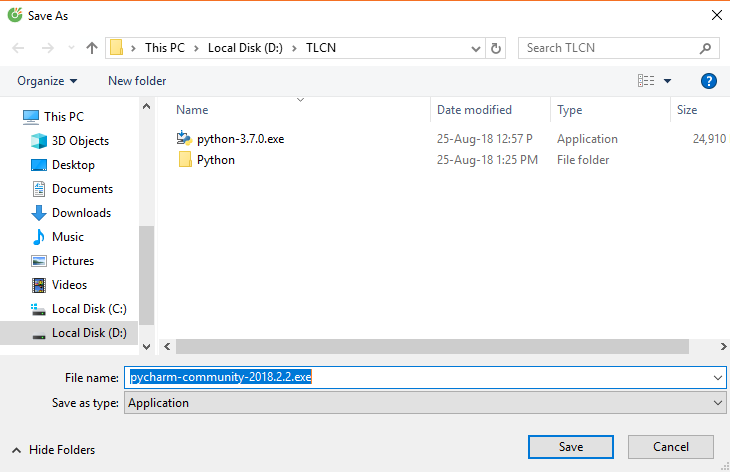
Sau đó chúng ta lưu file này vào thư mục **D:\TLCN** và mở teminal (**Windows + R** và gõ **cmd**)



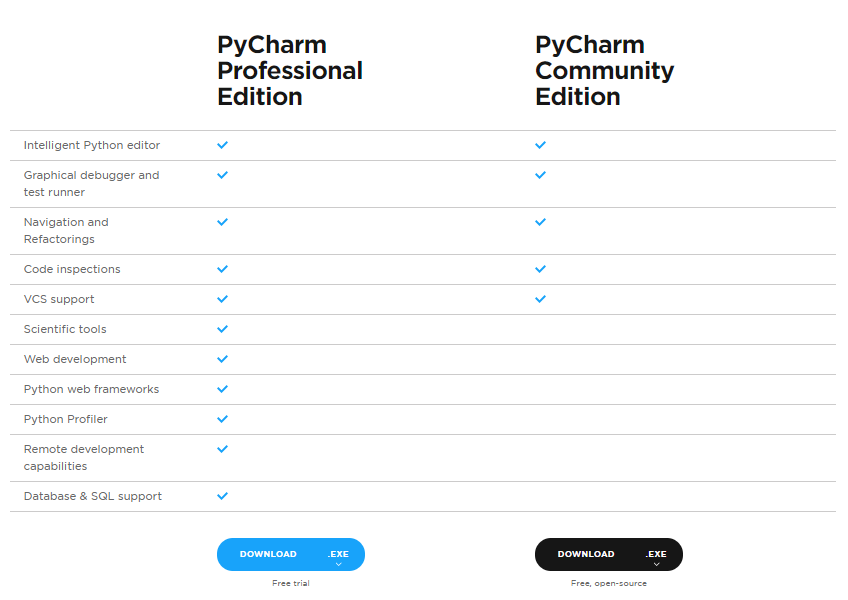
## 2.2 Cài đặt PyCharm cho Python

Chúng ta vào: <https://www.jetbrains.com/pycharm/download/#section=windows> để tải PyCharm.

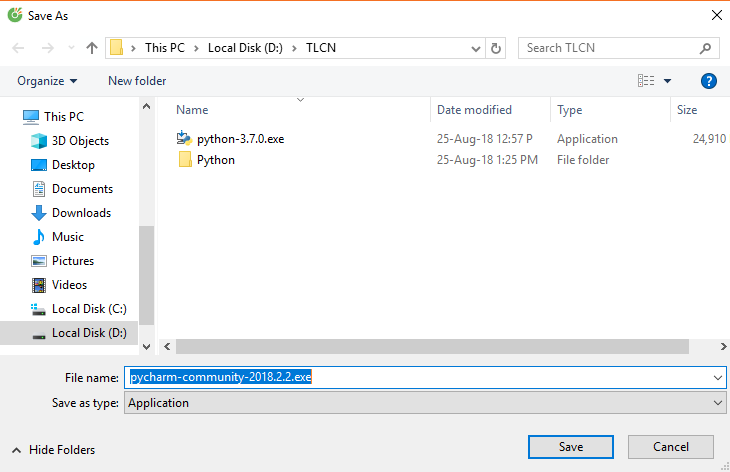




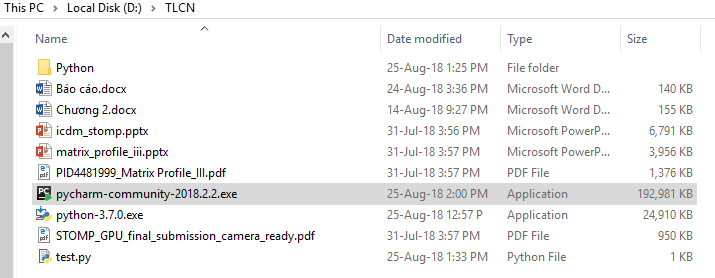
Với hình trên chúng ta thấy có 2 offer là **Professional** và **Community.** Với **Professional** là phiên bản đầy đủ có tính phí (miễn phí 30 ngày dùng thử) và **Community** là phiên bản miễn phí nhưng không có những tính năng để lập trình web



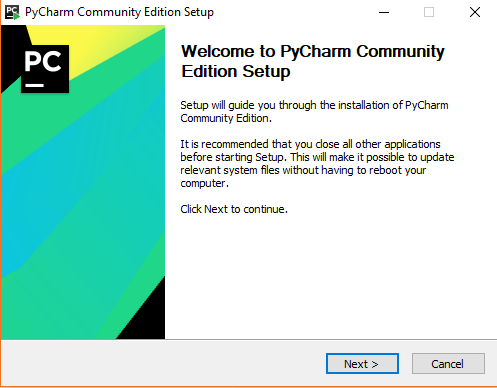
Sau khi click vào button **Download Community** thì ta được sẽ hiện ra thư mục để chúng ta lưu tệp **pycharm-community-2018.2.2.exe**

****

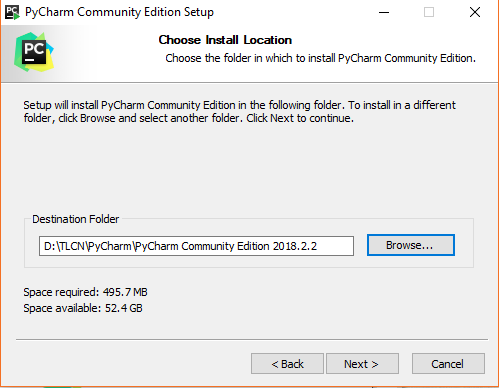
Sau khi tải xong thì chúng ta sẽ có tệp **pycharm-community-2018.2.2.exe** có dung lượng hơn 192 MB

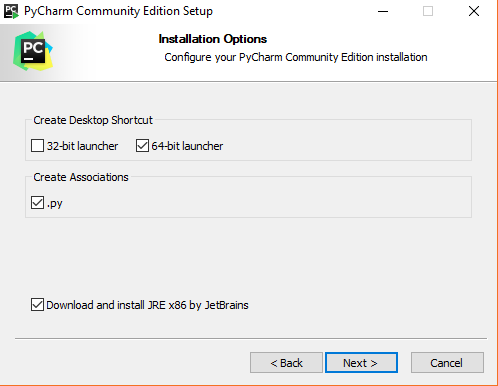


Dưới đây là giao diện chúng ta chạy tệp **pycharm-community-2018.2.2.exe** sau đó click **Next.**

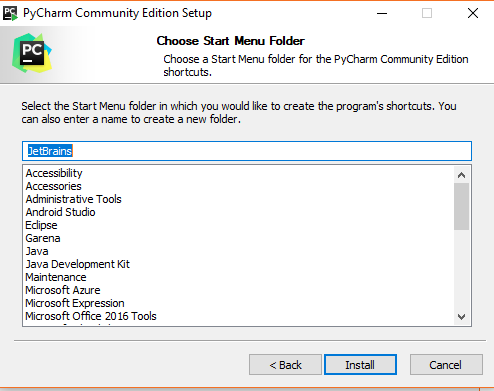


Cấu hình như hình bên dưới rồi click **Next.**

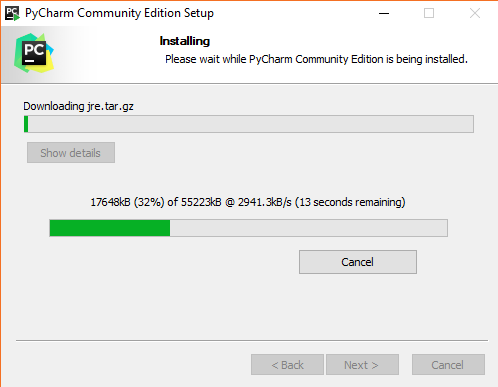


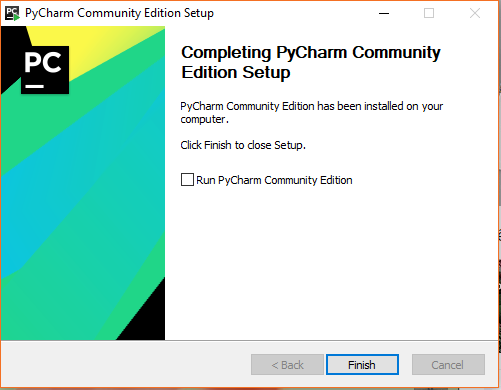


Cấu hình như hình dưới và click **Install.**

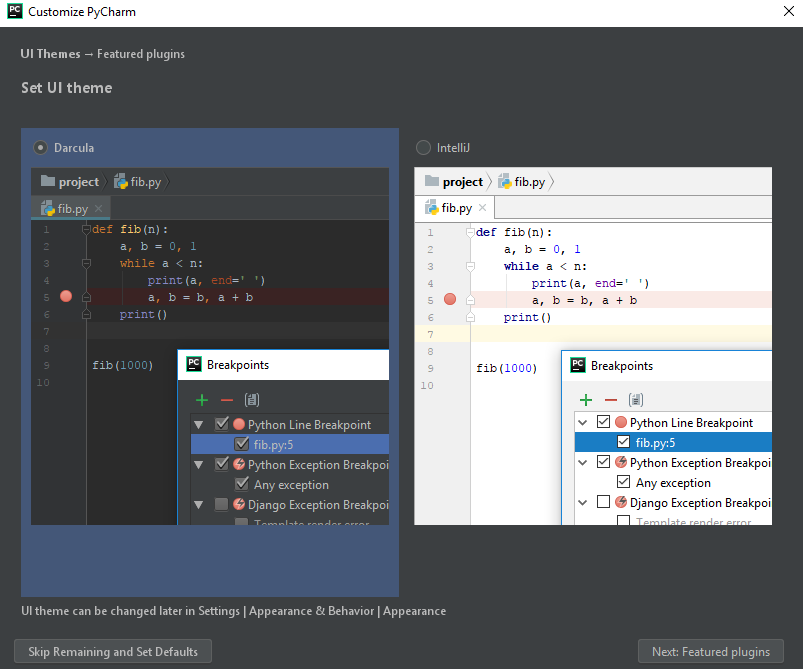


Chúng ta chờ nó tải xong là xong việc cài đặt PyCharm cho Python

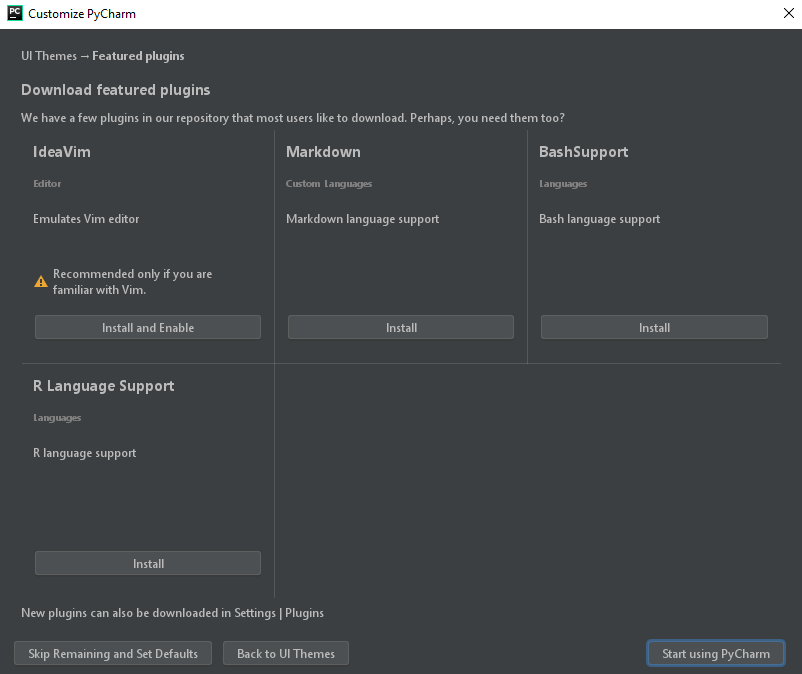




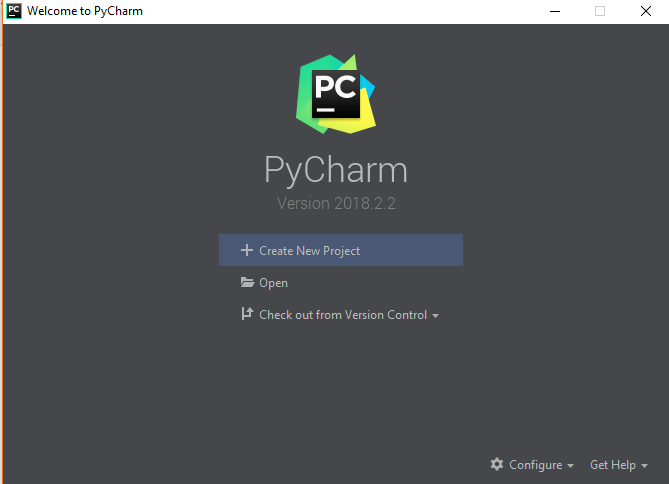
Sau khi cài xong chúng ta vào kiểm tra xem PyCharm có chạy được chưa. Khởi dộng PyCharm mới cài xong.



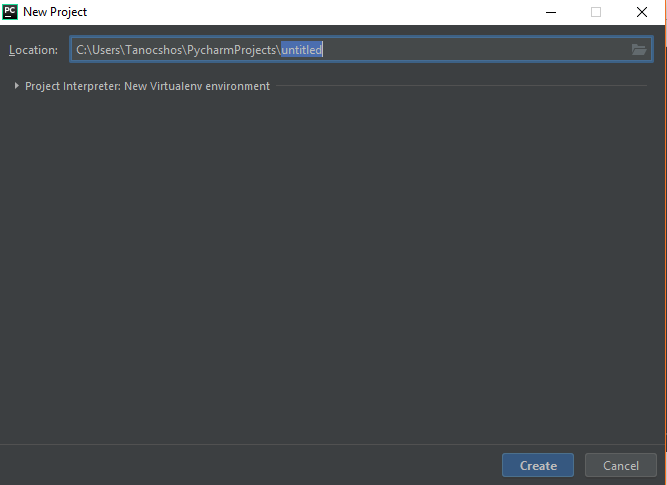
Chúng ta chọn kiểu theme mà chúng ta cảm thấy hợp nhất



Dưới đây là giao diện PyCharm đã hoàn tất.



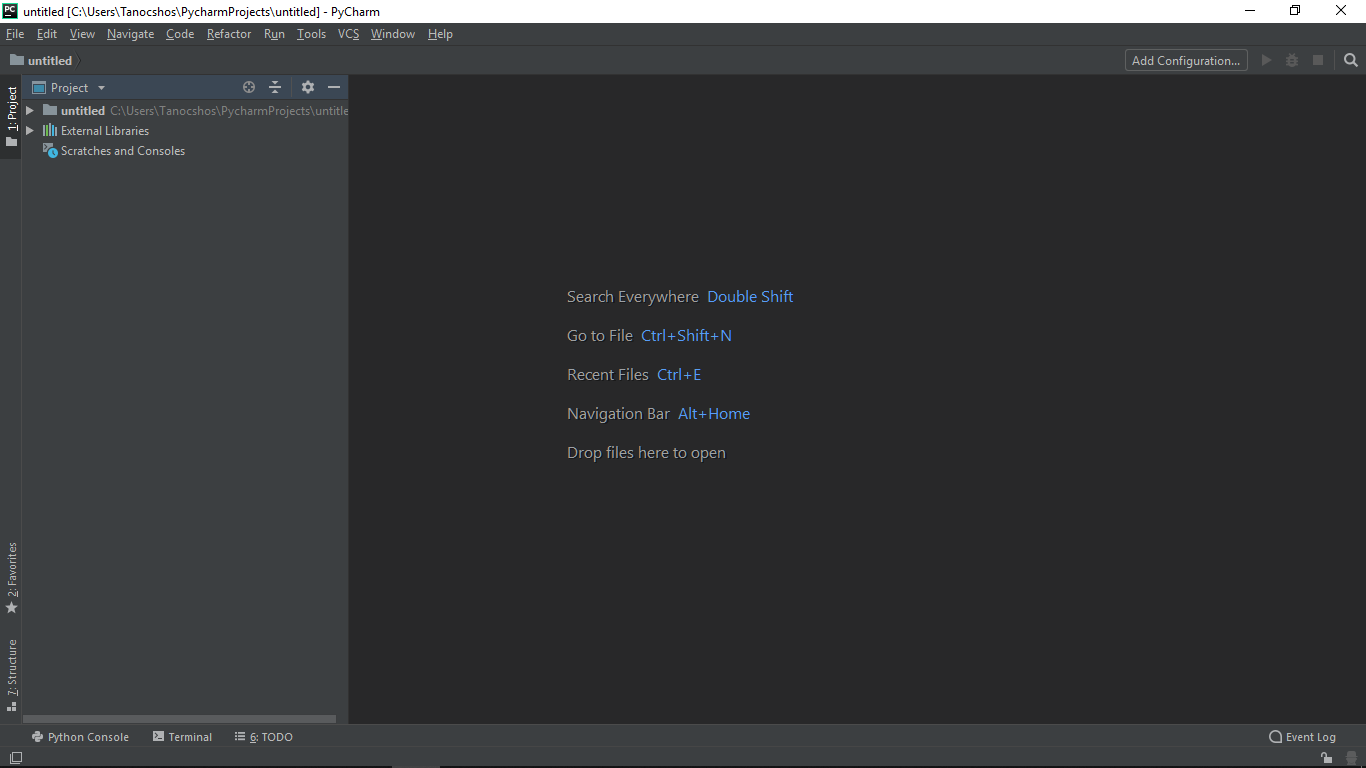
Hình trên chúng ta click **Create New Project.**



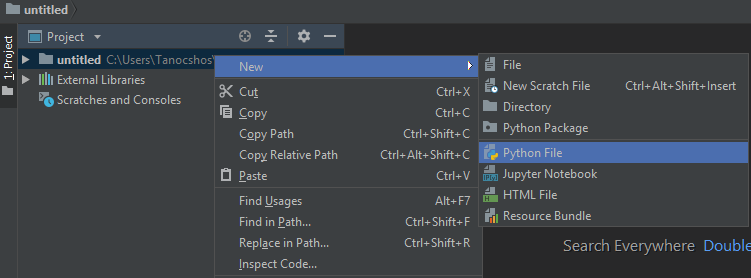
**Location :** Chọn nơi lưu trữ Projecct

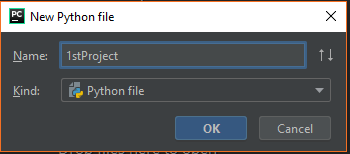
**Create:** Nhấn vào để tạo Project

Sau khi click Create, PyCharm sẽ tạo cấu trúc Project mặc định ban đầu như sau:

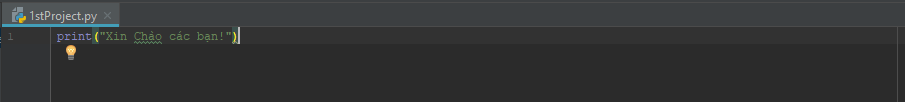


Để tạo tập tin Python: Ta bấm chuột phải vào Project **untitled /New/ Python File**:

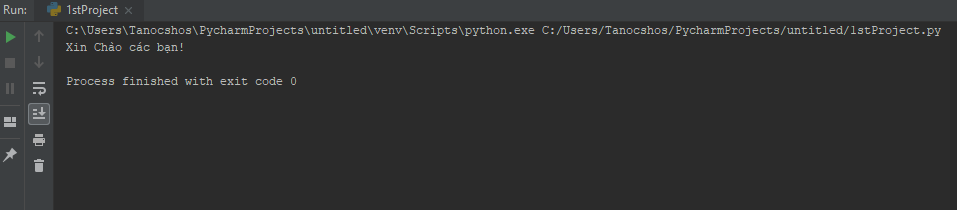




Chúng ta đặt tên file như trong hình rồi click **OK**.



Trong file **1stProject.py** chúng ta gõ dòng lệnh **print (“Xin chào các bạn!”)** và chạy thử xem kết quả.



Đây là kết quả chúng ta chạy thử thành công.

1. **Xử lý hàm trong Python**
2. Cú pháp

**def tenham(cac\_tham\_so)**

**“function\_docstring”**

**function\_suite**

**return [bieu\_thuc]**

1. Hàm return

Hàm return(bieu\_thuc) được sử dụng để gửi điều khiển quay trở lại người gọi với bieu\_thuc đã cho. Trong trường hợp không cung cấp bieu\_thuc, thì hàm return này sẽ trả về None. Nói cách khác, lệnh return được sử dụng để thoát khỏi định nghĩa hàm

1. Hàm với tham số thay đổi trong Python

* Có lúc có thể cần xử lý một hàm mà có số tham nhiều hơn là đã xác định trong khi định nghĩa hàm. Những tham số này được gọi là các tham số có số tham số thay đổi (variable-length args) và không được đặt tên trong định nghĩa hàm, không giống như các tham số bắt buộc và tham số mặc định.
* Cú pháp cho một hàm có số thay đổi là:

**def tenham([tham\_so\_chinh\_thuc,] \*var\_args\_tuple ):**

**"function\_docstring"**

**function\_suite**

**return [bieu\_thuc]**

* Một dấu \* được đặt trước tên biến để giữ các giá trị của các tham số loại này. Tuple này vẫn là trống nếu không có tham số bổ sung nào được xác định trong khi gọi hàm

1. Hàm vô danh trong Python

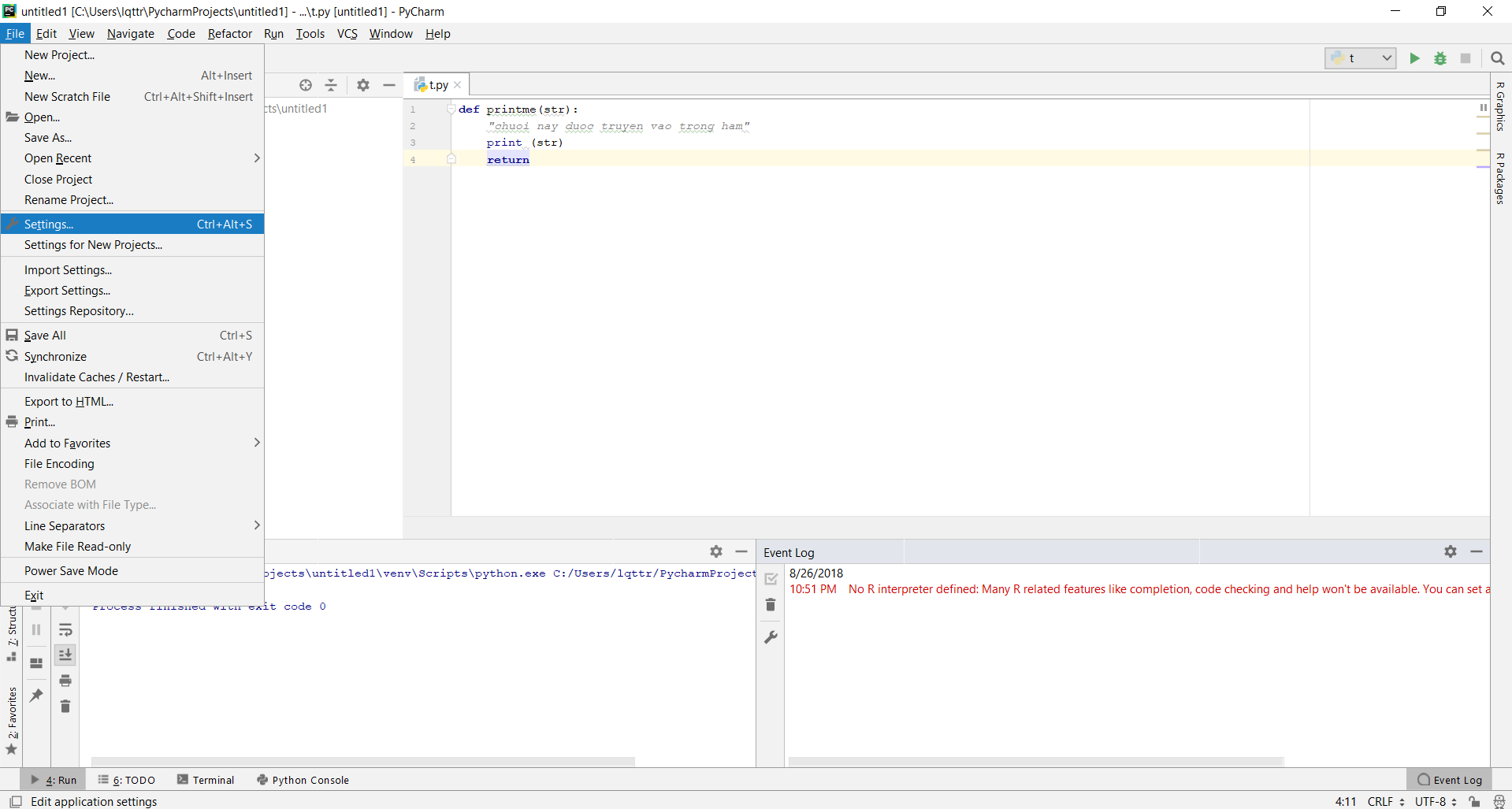
* Hàm vô danh là hàm không có tên và chúng không được khai báo theo cách chính thức bởi từ khóa def. Để khai báo hàm này, sử dụng từ khóa lambda. Lambda nhận bất kỳ lượng tham số nào và chỉ trả về một giá trị trong dạng một biểu thức đã được ước lượng. Không thể gọi trực tiếp gọi hàm vô danh để in bởi vì lambda cần một biểu thức. Ngoài ra, các hàm lambda có namespace cục bộ của chúng. Dưới đây là cú pháp của hàm lamda:

**lambda [arg1 [,arg2,.....argn]]:bieu\_thuc**

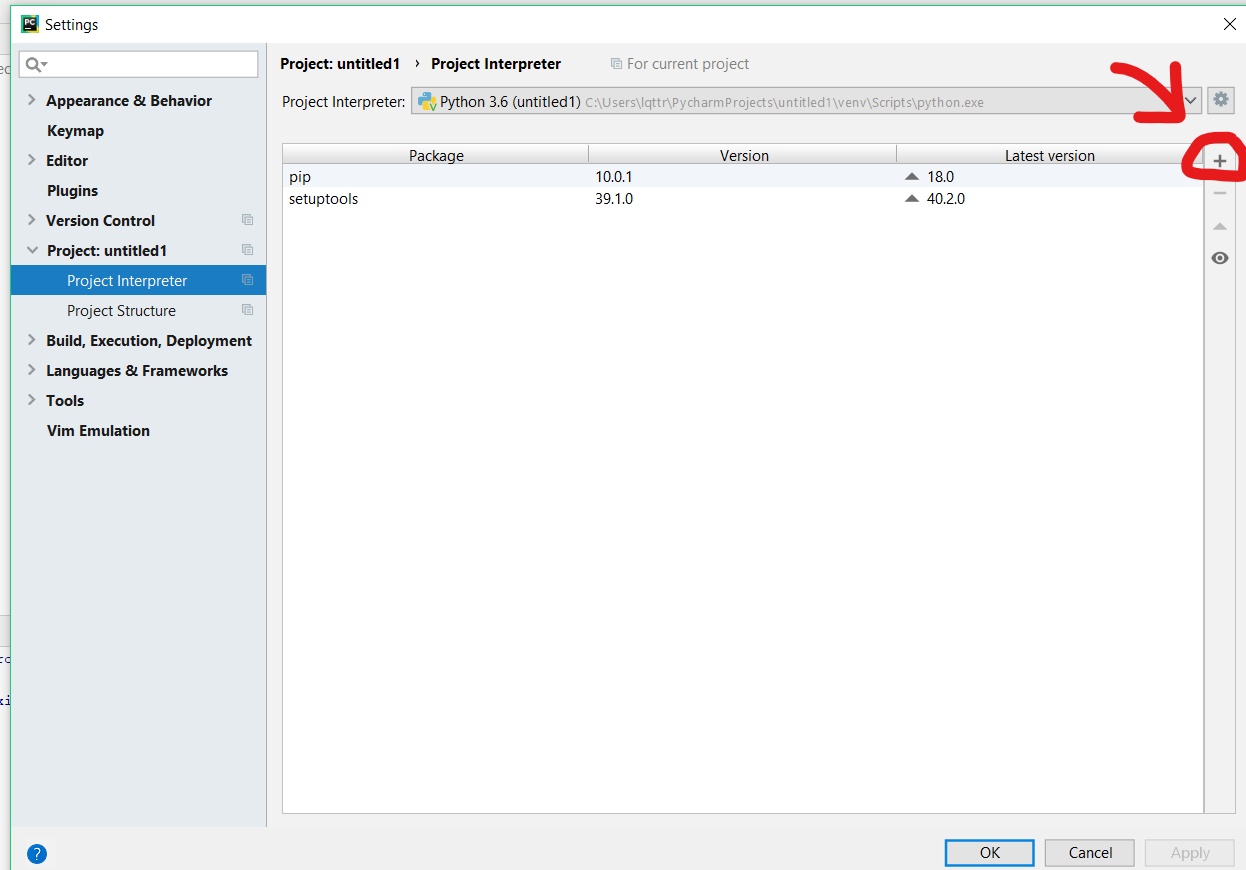
* Hàm vô danh được tạo bởi sử dụng từ khóa lambda, không phải bởi từ khóa def. Hàm này chỉ trả về một giá trị dưới dạng một biểu thức đã được ước lượng.

1. **Cách cài đặt thư viện cho Python**

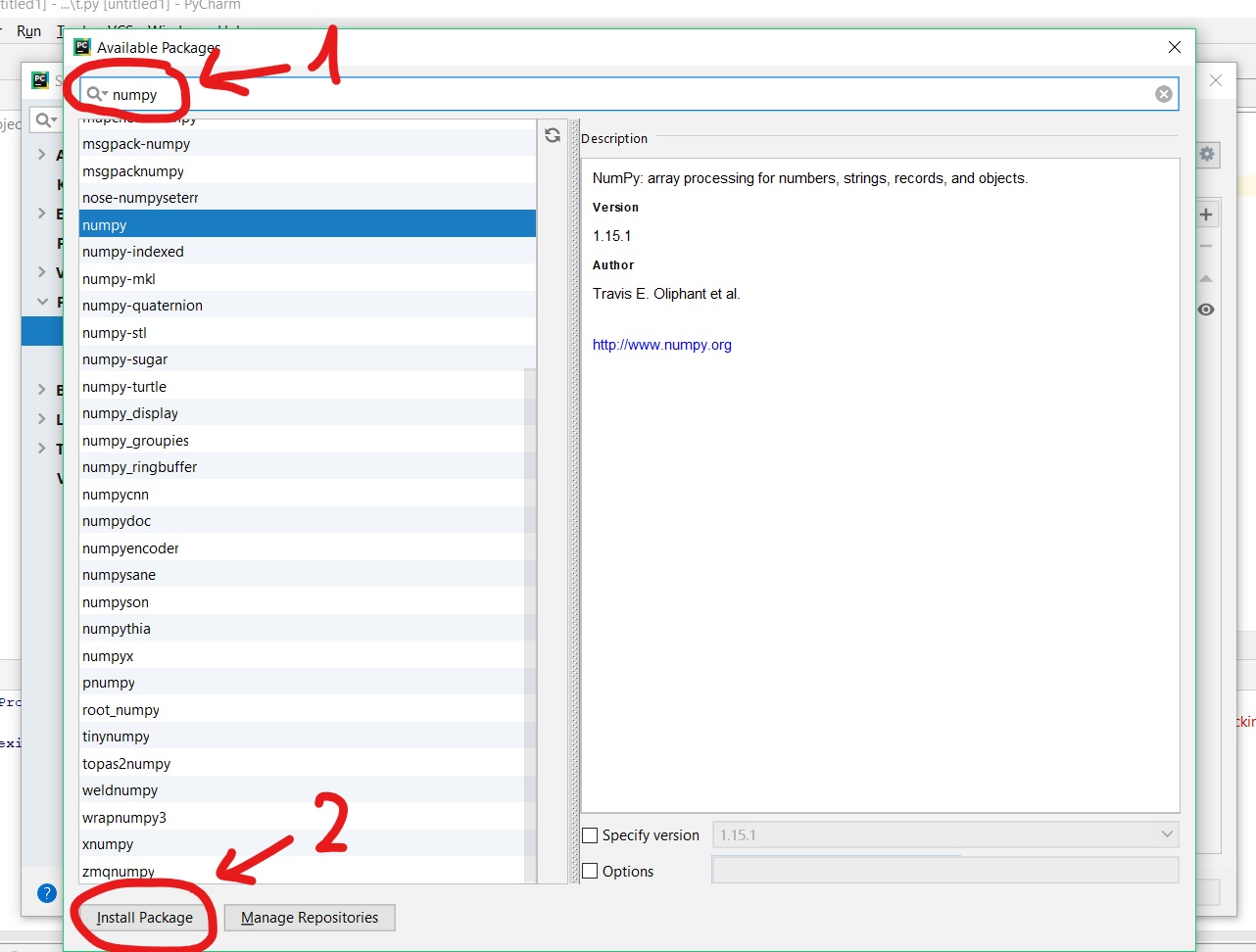
Đầu tiên từ Pycharm, ta sẽ vào File/Setting (Ctrl + Alt + S)



Sau đó, ta sẽ vào Project/ Project Interpreter, và nhấn vào dấu cộng như hình:



Cuối cùng, ta sẽ gõ tên thư viện cần thêm, sau đó chọn Install Package, đợi giây lát thì thư viện sẽ được cài đặt, và có thể sử dụng.



Tham khảo

[1] <https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>

1. **PHƯƠNG PHÁP PHÂN LỚP DỰA TRÊN LUẬT KẾT HỢP DỰ BÁO**
2. **Luật kết hợp trong Data mining**
3. **Một số khái niệm**

* Item (phần tử): là cặp thuộc tính hay giá trị.
* Itemset: một tập hợp các mục, chẳng hạn chuối, táo… xuất hiện thường xuyên với nhau trong 1 tập dữ liệu giao dịch.
* Ví dụ: I = {A, B, C}

Trong đó, I: itemset

A, B, C: items

* Transaction (giao dịch): là một tập T gồm các phần tử được giao dịch.
* Sequential pattern: là một chuỗi, chẳng hạn như mua một máy tính đầu tiên, sau đó một máy ảnh kỹ thuật số, và sau đó một thẻ nhớ, nếu nó xảy ra thường xuyên trong một cơ sở dữ liệu lịch sử mua sắm.
* Khai thác các mục thường xuyên dẫn đến việc khám phá các mối liên quan và mối tương quan giữa các mục trong các tập dữ liệu giao dịch hoặc quan hệ lớn.
* Một ví dụ điển hình về khai thác các mặt hàng thường xuyên là phân tích giỏ hàng.
* Quy trình phân tích giỏ hàng trên thị trường này phân tích thói quen mua hàng của khách hàng bằng cách tìm các liên kết giữa các mục khác nhau mà khách hàng đặt trong “giỏ mua sắm” của họ.

1. **Nội dung cơ bản:**

Cho cơ sở dữ liệu gồm các giao dịch T là tập các giao dịch t1, t2, …, tn.

* **T = {t1, t2, …, tn}. T gọi là cơ sở dữ liệu giao dịch (Transaction Database)**
* Mỗi giao dịch ti bao gồm tập các đối tượng I (gọi là itemset)
* **I = {i1, i2, …, im}.**Một itemset gồm k items gọi là k-itemset
* Mục đích của luật kết hợp là tìm ra sự kết hợp (association) hay tương quan (correlation) giữa các items. Những luật kết hợp này có dạng**X =>Y**
* Trong phân tích giỏ hàng, luật kết hợp X =>Y có thể hiểu rằng những người mua các mặt hàng trong tập X cũng thường mua các mặt hàng trong tập Y. (X và Y gọi là itemset).
* Ví dụ, nếu X = {Apple, Banana} và Y = {Cherry, Durian} và ta có luật kết hợp X =>Y thì chúng ta có thể nói rằng những người mua Apple và Banana thì cũng thường mua Cherry và Durian.
* X được xem là biến độc lập (Independent variable) còn Y được xem là biến phụ thuộc (Dependent variable)
* Độ hỗ trợ (Support) và độ tin cậy (Confidence) là 2 tham số dùng để đo lường luật kết hợp.
* Độ hỗ trợ (Support) của luật kết hợp X =>Y là tần suất của giao dịch chứa tất cả các items trong cả hai tập X và Y. Ví dụ, support của luật X =>Y là 5% có nghĩa là  5% các giao dịch X và Y được mua cùng nhau.
* Công thức để tính support của luật X =>Y như sau:
* http://bis.net.vn/photos/storage/image0011.png
* Trong đó: N là tổng số giao dịch.
* Độ tin cậy (Confidence) của luật kết hợp X =>Y là xác suất xảy ra Y khi đã biết X. Ví dụ độ tin cậy của luật kết hợp {Apple} =>Banana} là 80% có nghĩa là 80% khách hàng mua Apple cũng mua Banana.
* Công thức để tính độ tin cậy của luật kết hợp X =>là xác suất có điều kiện Y khi đã biết X như sau :
* http://bis.net.vn/photos/storage/image005.gif
* Trong đó: n(X) là số giao dịch chứa X

Ví dụ:

|  |  |
| --- | --- |
| TID | Transaction |
| 1 | {A; B; C} |
| 2 | {A; C;D;E} |
| 3 | {A; B; E} |
| 4 | {A; C; E} |

Sup(A) = \* 100% = 100%

Sup(AB) = \* 100% = 50%

Conf(AB) = \* 100% = 50%

* Frequent itemset (tập phần tử phổ biến): là tập phần tử có độ hỗ trợ tối thiểu là độ hỗ trở nhỏ nhất.

Ví dụ: Sup(I) minsup thì I là tập phần tử phổ biến.

* Strong association rule (luật kết hợp mạnh):
* Luật kết hợp có độ hỗ trợ và độ tin cậy thỏa minsup và minconf.

Ví dụ: Cho luật kết hợp AB

Sup(AB) minsup,

Conf(AB) minconf thì luật AB được giữ lại

1. **Thuật toán Apriori**

* Duyệt toàn bộ dữ liệu trong giao dịch để có được độ hỗ trợ S của 1-itemset, so sánh S với min\_sup, để có được 1-itemset (L1)
* Sử dụng Lk-1 nối Lk-1  để sinh ra ứng viên k-itemset. Loại bỏ các itemsets không phải là frequent itemsets thu được k-itemset
* Duyệt toàn bộ dữ liệu trong giao dịch để có được độ hỗ trợ của mỗi ứng viên k-itemset, so sánh S với min\_sup để thu được frequent k –itemset (Lk)
* Lặp lại từ bước 2 cho đến khi ứng viên set (C) trống (không tìm thấy frequent itemsets)
* Với mỗi frequent itemset I, sinh tất cả các tập con s không rỗng của I
* Với mỗi tập con s không rỗng của I, sinh ra các luật  **s => (I-s)** nếu độ tin cậy (Confidence)  của nó > =min\_conf

1. **Bài toán phân lớp trong data mining**

Phân lớp là một dạng phân tích dữ liệu chiết xuất các mô hình mô tả các lớp dữ liệu quan trọng. Các mô hình như vậy, được gọi là các trình phân lớp, dự đoán các lớp nhãn phân lớp.

Đầu vào: một tập các mẫu dữ liệu huấn luyện, với một mẫu phân lớp cho mỗi mẫu dữ liệu.

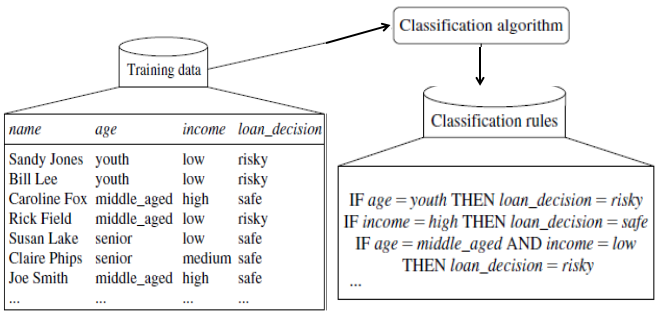
Đầu ra: mô hình dựa trên tập huấn luyện và những nhãn phân lớp.

\*Cách tiếp cận chung của phân lớp

Phân lớp dữ liệu là một quá trình gồm hai bước: giai đoạn học tập/đào tạo (một mô hình phân lớp được xây dựng) và một bước phân lớp (nơi mô hình được sử dụng để dự đoán các nhãn lớp cho dữ liệu đã cho).

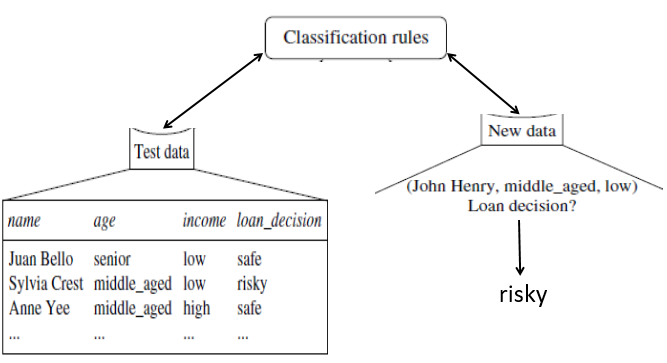
* Giai đoạn học tập/đào tạo:
  + Mỗi bộ/mẫu dữ liệu được phân vào mỗi lớp được xác định trước.
  + Lớp của một bộ/mẫu dữ liệu được xác định bởi thuộc tính gán nhãn lớp.
  + Tập các bộ/mẫu dữ liệu huấn luyện - tập huấn luyện - được dùng để xây dựng mô hình.
  + Mô hình được biểu diễn bởi các luật phân lớp, các cây quyết định hoặc các công thức toán học.

Ví dụ: Giai đoạn học tập/ đào tạo.



* Giai đoạn phân lớp:
  + Phân lớp cho những đối tượng mới hoặc chưa được phân lớp.
  + Đánh giá độ chính xác của mô hình.

Ví dụ: Giai đoạn phân lớp.



Có hai loại phân lớp:

* Học tập được giám sát: nhãn lớp của mỗi bộ huấn luyện được cung cấp.
* Học tập không được giám sát (hoặc phân nhóm): nhãn lớp của mỗi bộ huấn luyện không được biết, và số hoặc tập dữ liệu các lớp được học có thể không được biết trước.

Tập dữ liệu:

* Tập dữ liệu huấn luyện: được sử dụng để xây dựng trình phân loại
* Tập dữ liệu kiểm tra: gồm các bộ kiểm tra và các nhãn lớp liên quan của chúng và được sử dụng để ước tính độ chính xác, dự báo của một bộ phân loại. Nó phải độc lập với tập dữ liệu huấn luyện.

Độ chính xác của một trình phân lớp trên một tập dữ liệu kiểm tra đã cho là phần trăm các bộ kiểm tra được phân lớp chính xác bởi trình phân lớp.

Nếu độ chính xác của trình phân lớp được coi là chấp nhận được, thì trình phân lớp có thể được sử dụng để phân lớp các bộ dữ liệu trong tương lai mà nhãn lớp không được biết.

1. **Phân lớp dựa trên luật kết hợp dự báo[2]**
2. **Tóm tắt**

* Các nghiên cứu gần đây về khai phá dữ liệu đã đề xuất một cách phân loại mới, được gọi là phân lớp kết hợp, theo một số báo cáo, thì nó đạt được độ chính xác cao hơn so với các phương pháp phân loại truyền thống như thuật toán phân lớp C4.5. Tuy nhiên, cách này cũng xuất phát từ ​​hai yếu tố chính: (1) nó tạo ra một số lượng rất lớn các quy tắc kết hợp, dẫn đến chi phí xử lý cao; và (2) biện pháp đánh giá quy tắc dựa trên nó có thể dẫn đến tình trạng quá tải.
* So với phân lớp kết hợp, các phân lớp dựa trên quy tắc truyền thống, chẳng hạn như C4.5, FOIL và RIPPER, nhanh hơn đáng kể nhưng độ chính xác của chúng, trong hầu hết các trường hợp, có thể không cao. Và 1 cách phân loại mới là CPAR (Classiﬁcation based on Predictive Association Rules), kết hợp các ưu điểm của cả phân lớp kết hợp và phân loại dựa trên quy tắc truyền thống. Thay vì tạo ra một số lượng lớn các quy tắc ứng cử viên như trong phân lớp kết hợp, CPAR sử dụng một thuật toán tham lam để tạo ra các quy tắc trực tiếp từ dữ liệu huấn luyện. Hơn nữa, CPAR tạo ra và kiểm tra nhiều quy tắc hơn dựa trên quy tắc truyền thống để tránh thiếu các quy tắc quan trọng. Để tránh quá tải, CPAR sử dụng độ chính xác mong đợi để đánh giá từng quy tắc và sử dụng các quy tắc k tốt nhất trong dự đoán.

1. **Giới thiệu**

* Trong những năm gần đây, một phương pháp mới được gọi là phân lớp kết hợp được đề xuất để tích hợp khai thác quy tắc kết hợp và phân loại. Nó sử dụng thuật toán khai phá quy tắc kết hợp, chẳng hạn như Apriori hoặc FPgrowth, để tạo ra bộ quy tắc kết hợp hoàn chỉnh. Sau đó, nó chọn một bộ quy tắc chất lượng cao và sử dụng quy tắc này được đặt để dự đoán. Các thí nghiệm cho thấy rằng phương pháp này đạt được độ chính xác cao hơn các phương pháp tiếp cận phân loại truyền thống như C4.5. Tuy nhiên, phân loại kết hợp sẽ không hiệu quả do thực tế là nó thường tạo ra một số lượng lớn các quy tắc trong khai phá quy tắc kết hợp, và cũng cần phải có các quy tắc để chọn các quy tắc chất lượng cao trong số đó.
* CPAR kế thừa ý tưởng cơ bản về FOIL trong việc tạo quy tắc và tích hợp các tính năng của phân lớp kết hợp trong phân tích quy tắc dự đoán. So với phân lớp kết hợp, CPAR có những ưu điểm sau: (1) CPAR tạo ra một bộ quy tắc tiên đoán chất lượng cao hơn nhiều trực tiếp từ tập dữ liệu; (2) để tránh tạo ra các quy tắc dư thừa, CPAR tạo ra mỗi quy tắc bằng cách xem xét bộ quy tắc "đã được tạo"; và (3) khi dự đoán nhãn lớp của một ví dụ, CPAR sử dụng các quy tắc k tốt nhất cho ví dụ này.
* Hơn nữa, CPAR sử dụng các tính năng sau để nâng cao hơn nữa tính chính xác và hiệu quả của nó: (1) CPAR sử dụng lập trình động để tránh tính toán lặp lại trong quá trình tạo quy tắc; và (2) khi tạo ra các quy tắc, thay vì chỉ chọn chữ tốt nhất, tất cả các chữ cái gần nhất để lựa chọn để các quy tắc quan trọng sẽ không bị bỏ qua.
* CPAR tạo ra một bộ quy tắc nhỏ hơn, với chất lượng cao hơn và dự phòng thấp hơn so với phân loại kết hợp. Kết quả là, CPAR có nhiều thời gian hơn trong cả việc tạo quy tắc và dự đoán nhưng đạt được độ chính xác cao như phân lớp kết hợp.

1. **Phân lớp dựa trên luật**

* Cho T là một bộ các tập dữ liệu. Mỗi bộ t trong T theo sau lược đồ (A1, A2, … , Ak), trong đó A1, A2, ... , Ak là k thuộc tính. Mỗi thuộc tính liên tục được phân tách lần đầu tiên vào một thuộc tính phân lớp.
* Định nghĩa 2.1. (literal) một literal P là một cặp thuộc tính-giá trị, lấy dưới dạng (Ai, v), trong đó Ai là một thuộc tính và v một giá trị. Một bộ t thỏa mãn một literal p = (Ai, v) nếu và chỉ nếu ti = v, trong đó ti là giá trị của thuộc tính ith của t.
* Định nghĩa 2.2. (luật) một luật r, có dạng của "p1 ∧ p2 ∧ · · · ∧ pl → c," bao gồm sự kết hợp của literals p1, p2,. . . , pl, liên kết với một nhãn lớp c. Một bộ t thỏa mãn nội dung của luật r nếu và chỉ nếu nó thỏa mãn mỗi literal trong luật r. Nếu t thỏa mãn nội dung r, r dự đoán rằng t là của lớp c. Nếu một luật không có literal, nội dung của nó được thỏa mãn bởi bất kỳ bộ dữ liệu nào..
* Hai cái phân lớp trên được tổng hợp từ CBA và CMAR. CBA đầu tiên tạo ra tất cả các luật liên kết với độ hỗ trợ và độ tin cậy nhất định như các luật ứng viên. Sau đó, nó chọn một bộ luật nhỏ từ chúng để tạo thành một trình phân lớp. Khi dự đoán nhãn lớp cho ví dụ, luật tốt nhất (tức là, với độ tin cậy cao nhất) mà nội dung được thỏa mãn bởi ví dụ được chọn để dự đoán.
* CMAR tạo và đánh giá các luật theo cách tương tự như CBA (nhưng sử dụng cấu trúc FPtree hiệu quả hơn). Một sự khác biệt lớn là nó sử dụng nhiều luật trong dự đoán, sử dụng trọng số χ 2. Các thí nghiệm của họ cho thấy rằng CMAR đạt được độ chính xác cao hơn CBA.
* Khi bộ dữ liệu chứa một số lượng lớn các hàng và/hoặc cột, cả việc tạo luật và lựa chọn luật trong CBA và CMAR đều tốn thời gian. Trong bài báo này, một cách tiếp cận mới được đề xuất để khắc phục vấn đề này: Thay vì tạo ra các luật ứng cử viên, một bộ luật dự báo nhỏ được tạo trực tiếp từ tập dữ liệu dựa trên dự đoán luật và phân tích vùng phủ sóng.

1. **Thuật toán FOIL**

* FOIL được đề xuất và phát triển bởi Quinlan (Quinlan, 1990). FOIL học các tập dữ liệu chỉ bao gồm 2 lớp, lớp các ví dụ “tích cực” và ví dụ “tiêu cực”. FOIL học mô tả lớp với lớp “tích cực”.
* Đầu vào của Foil gồm các tiền điều kiện và các kết luận.
* Đầu ra là một tập các luật sinh từ các tiền điều kiện và các kết luận đó.
* Mỗi bước Foil sẽ thêm một literal vào các tiền điều kiện của luật đang huấn luyện. Thuật toán sử dụng hàm Foil\_Gain để tính toán lựa chọn một literal trong tập các literal ứng cử.
* Trong Foil có hai trạng thái chính :
* Separate stage (trạng thái phân tách) : Bắt đầu một trạng thái mới
* Conquer State (trạng thái chế ngự): Kết hợp các literal để xây dựng thân của mệnh đề.
* Pha “tách rời” của thuật toán bắt đầu từ luật mới trong khi pha “chế ngự” xây dựng 1 liên kết các literal làm thân của luật. Mỗi luật mô tả 1 tập con nào đó các ví dụ ”tích cực” và không có ví dụ tiêu cực. FOIl có 2 toán tử: bắt đầu một luật mới với thân luật rỗng và thêm một literal để kết thúc luật hiện tại. FOIL kết thúc việc bổ sung literal khi không còn ví dụ tiêu cực được bao phủ bởi luật và bắt đầu luật mới đến khi tất cả mỗi ví dụ tích cực được bao phủ bởi một luật nào đó.
* Các ví dụ tích cực được phủ bởi mệnh đề sẽ được tách ra khỏi tập dạy, quá trình sẽ được tiếp tục để học các mệnh đề tiếp theo với các ví dụ còn lại, nó sẽ kết thúc khi không có các ví dụ tích cực thêm nữa.
* Thuật toán:
* Gọi Pos là tập các ví dụ tích cực
* Gọi Neg là tập các ví dụ tiêu cực
* Đặt NewClauseBody bằng rỗng
* Trong khi Pos chưa rỗng thực hiện:

Separate: bắt đầu một luật mới

* Loại khỏi Pos tất cả những ví dụ thoả mãn NewClauseBody.
* Đặt lại Neg là tập các ví dụ tiêu cực ban đầu
* Đặt lại NewClauseBody bằng rỗng

Trong khi Neg chưa rỗng thực hiện.

Conquer (Xây dựng thân mệnh đề)

* Chọn Literal L
* Kết hợp vào NewClauseBody.
* Loại khỏi Neg những ví dụ mà không thoả mãn L.
* FOIL sử dụng giải thuật leo đồi để bổ sung các literal với thông tin thu được lớn nhất vào một luật. Với mỗi biến đổi của một khẳng định P, FOIL đo lượng thông tin đạt được. Để lựa chọn literal với thông tin đạt được cao nhất, nó cần biết bao nhiêu bộ tích cực và tiêu cực hiện tại để đảm bảo bởi các biến đổi của mỗi khẳng định được xác định theo cách dàn trải.
* Công thức tính infortmaion gain của Foil là:

Gain(Literal)=T++ \*(log2(P1/P1+N1) - log2(P0/P0+N0))

P0 và N0 là số ví dụ dương và âm trước khi thêm một literal L vào mệnh đề

P1 và N1 là số ví dụ dương và âm sau khi thêm literal L vào mênh đề.

T++ là số ví dụ dương cố định cả trước và sau khi thêm literal .(nghĩa là số ví dụ đúng với cả hai luật R và R’\_là R sau khi đã thêm vào literal L)

1. **CPAR[1]**
2. **Mục tiêu**

* Đầu vào: tập dữ liệu huấn luyện và tập dữ liệu kiểm tra có dán nhãn
* Đầu ra: dự đoán nhãn và tính độ chính xác của tập dữ liệu kiểm tra

1. **Phương pháp thực hiện**

* Thuật toán CPAR (Classification based on Predictive Association Rules-Phân lớp dựa trên luật kết hợp dự doán) (Xiaoxin Yin và Jiawei Han) là một phần mở rộng của PRM mà PRM là một phần mở rộng của FOIL. Sự khác biệt giữa CPAR và PRM là thay vì chỉ chọn thuộc tính hiển thị là gain tốt nhất trên mỗi lần lặp (như trong FOIL và PRM), CPAR có thể chọn một số thuộc tính nếu các thuộc tính đó tương tự gain tốt nhất. Điều này được thực hiện bằng cách tính toán độ gain đầu tiên và áp dụng GAIN\_SIMILARITY\_RATIO cho điều này (Xiaoxin Yin và Jiawei Han đề xuất 0,99). Tất cả các thuộc tính với gain tốt hơn: **bestGain\*GAIN\_SIMILARITY\_RATIO**
* Sau đó độ gain tốt nhất đó được chọn để xử lý tiếp.
* CPAR (như FOIL và PRM) lấy dữ liệu đầu vào là một bộ dữ liệu có giá trị nhị phân và tạo ra một tập các CARs (Classification Association Rules). Kết quả phân lớp theo thuật toán PRM bao gồm một danh sách liên kết các luật được sắp xếp theo độ chính xác Laplace của các luật riêng lẻ.

method: startCPAR

Parameters: none

Global access to: R, C

----------------------------------------------

generate an empty global attributes array A

For each c in C

generate global P and N example arrays

generate global PN array

determine minimum total Weight threshold

while (total weight P > minimum total Weight threshold)

A' 🡨 A, N' 🡨 N, P' 🡨 P, PN' 🡨 PN

if no attributes exist with weightings that can

produce a gain above minimum break

cparGeneration({},c)

end loop

end loop

Hình 1: Phương thức startCPAR

Trong đó:

* R: là tập dữ liệu nhị phân
* C: là tập hợp các nhãn lớp
* c: là giá trị mỗi nhãn
* P: mảng chứa các mẫu tin thuộc mẫu tích cực
* N: mảng chứa các mẫu tin thuộc mẫu tiêu cực
* PN: mảng chứa trọng số của mảng P và mảng N
* A: mảng rỗng chứa các giá trị gain

method: cparGenerastion/2

Parameters: parameters ante (antecedent so far) and

cons (consequent) for current rule.

Global access to: A, A', N, N', P, P', PN, PN'

----------------------------------------------

for each a in A

if (a not in antecedent1) calculated gain using

information in PN array and add to attribute array

end loop

i = "available" column in A with best gain

if (A[i][0] <= MIN\_BEST\_GAIN) return

loop through attribute array and find attribute a' with best gain

if (best gain <= MIN\_BEST\_GAIN) {

add antecedent🡪c to rule list

for all records in P reduce weighting by decay factor

and adjust PN array accordingly

return

gainThreshold = bestGain\*GAIN\_SIMILARITY\_RATIO

for each a in A

if a available1 and a.gain>gainThreshold

tempP' 🡨 P', tempN' 🡨 N', tempA' 🡨 A', tempNP'🡨NP'

add a' to antecedent

remove examples from N' and P' that do not contain

antecedent and adjust NP array accordingly

if (N' == {})

add antecedent->c to rule list

for all roecords in P reduce wieighting by decay factor

and adjust NP array accordingly

else prmGeneration(antecedent,c)

P' 🡨 tempP, N' 🡨 tempN', A' 🡨 tempA', NP' 🡨 tempNP'

end loop

Hình 2: Phương thức cparGeneration

Trong các thí nghiệm của Xiaoxin Yin và Jiawei Han đã sử dụng một hằng số gain tối thiểu là 0.7, và decayFactor (hệ số phân rã) là 1/3, newWeighting được tính như sau:

**newWeighting = oldWeighting \* decayFractor**

và gain được tính như sau:

**WP = totalWeight (P)**

**WN = totalWeight (N)**

**WP'= totalWeight (P')**

**WN'= totalWeight (N')**

**gain (a) = WP'log (WP' / WP '+ NP') - log (WP / WP + NP)**

**Laplace = nc + 1/ntot + k**

Trong đó: k: số lớp

nc: tổng số bộ trong tập dữ liệu huấn luyện thỏa mãn tiền điều kiện và kết quả.

Ngưỡng trọng số tối thiểu cho P được tính bằng cách nhân trọng số bắt đầu của P với TOTAL\_WEIGHT\_THRESHOLD được đặt thành 0,05 trong khi thử nghiệm. GAIN\_SIMILARITY\_RATIO được đặt thành 0,99.

1. **Quy trình xử lý**

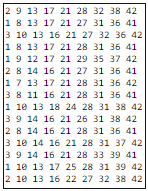
* Có thể sử dụng CPAR để tạo ra hai hoặc nhiều luật với tiền điều kiện và kết quả giống hệt nhau. Điều này là do các records không được loại bỏ khỏi các danh sách N và P khi một luật thỏa mãn chúng đã được tìm thấy, nhưng thay vào đó các records này chỉ đơn giản là giảm bớt sự sụt giảm của chúng. Việc giảm trọng số này tạo thành một phần của thuật toán để tính toán gain, tuy nhiên vẫn có thể cho các thuộc tính tương tự được chọn để tạo tiền điều kiện của một luật vì các thuộc tính này (mặc dù giảm trọng số) vẫn mang độ gain tốt nhất. Sau cùng, trọng số của các records được giảm cho đến việc các thuộc tính không tạo ra gain tốt nhất thì nó không được chọn. Trường hợp xảy ra này, các luật có độ chính xác thấp hơn sẽ bị xóa khỏi danh sách luật.

1. **Quá trình kiểm tra**

* Trong quá trình kiểm tra, đối với mỗi record trong bộ thử nghiệm (theo khuyến cáo trong Yin và Han 2003), record được phân lớp theo quy trình sau:
* Lấy tất cả các luật mà tiền điều kiện là một tập con của record đã cho.
* Chọn các luật K tốt nhất cho mỗi lớp theo độ chính xác Laplace của chúng. Trong các thí nghiệm K được đặt thành 5).
* Xác định độ chính xác trung bình trên các luật đã chọn cho mỗi lớp
* Chọn lớp có độ chính xác trung bình tốt nhất.

1. **Ví dụ**

* Ở ví dụ này, GAIN\_SIMILARITY\_RATIO đã được đặt thành 0,6 thay vì giá trị thông thường là khoảng 0,99.
* Dưới đây tập dữ liệu của Pima Idian (15 dòng)



* 9 cột tương ứng với 9 thuộc tính (theo thứ tự từ trái sang phải).

1. Mang thai: số lần mang thai.
2. Glucose: Nồng đồ glucose trong huyết tương trong 2 giờ thử nghiệm dung nạp glucose.
3. Huyết áp: Huyết áp tâm trương (mm/Hg)
4. Độ dày da (mm)
5. Insulin: huyết thanh insulin trong 2 giờ
6. Chỉ số khối cơ thể (BMI): nhỏ hơn 18.5 là gầy, 18.5 – 24.9 là bình thường, 25 – 29.9 là hơi béo, trên 30 là béo phì.
7. Chức năng phối hợp bệnh tiểu đường
8. Tuổi
9. Kết quả (41 là bệnh tiểu đường: nhóm tích cực/dương tính, 42 là không bị bệnh: nhóm tiêu cực/âm tính)

Ví dụ chạy tay thuật toán.

Giả sử ta có tập dữ liệu:

2 9 13 17 21 28 32 38 42

1 8 13 17 21 27 31 36 41

3 10 13 16 21 27 32 36 42

1 8 13 17 21 28 31 36 41

1 9 12 17 21 29 35 37 42

2 8 14 16 21 27 31 36 41

1 7 13 17 21 28 31 36 42

3 8 11 16 21 28 31 36 41

1 10 13 18 24 28 31 38 42

3 9 14 16 21 26 31 38 42

2 8 14 16 21 28 31 36 41

3 10 14 16 21 28 31 37 42

3 9 14 16 21 28 33 39 41

1 10 13 17 25 28 31 39 42

2 10 13 16 22 27 32 38 42

**Bước 1:**

generate an empty global attributes array A

Tạo mảng A chứa tập các thuộc tính từ 1 – 40, mỗi thuộc tính mang giá trị 0

**Bước 2:** Với giá trị mỗi lớp trong tập nhãn lớp thực hiện vòng lặp tạo mảng P, mảng N và mảng PN, xác định tổng trọng số ngưỡng thấp nhất.

For each c in C

generate global P and N example arrays

generate global PN array

determine minimum total Weight threshold

* Bắt đầu với lớp 41 và tạo các tập P và N và một mảng NP. Các số cột đầu tiên trong mảng NP là các nhãn thuộc tính (1 đến 40), hai cột sau (WP và WN). WP là tổng trọng số của thuộc tính tương ứng trên dòng trong bảng P và WN tổng trọng số của thuộc tính tương ứng trên dòng trong bảng N. Khi bắt đầu, mỗi thuộc tính có trọng số là 1 liên kết với nó, vì vậy thuộc tính 1 xảy ra hai lần trong P và bốn lần trong N.
* Mảng P (positiveExamples)

Trọng số

1 8 13 17 21 27 31 36 41 (1.0)

1 8 13 17 21 28 31 36 41 (1.0)

2 8 14 16 21 27 31 36 41 (1.0)

3 8 11 16 21 28 31 36 41 (1.0)

2 8 14 16 21 28 31 36 41 (1.0)

3 9 14 16 21 28 33 39 41 (1.0)

* Mảng N ( negativeExamples)

|  |
| --- |
| Trọng số  {2 9 13 17 21 28 32 38 42} (1.0)  {3 10 13 16 21 27 32 36 42} (1.0)  {1 9 12 17 21 29 35 37 42} (1.0)  {1 7 13 17 21 28 31 36 42} (1.0)  {1 10 13 18 24 28 31 38 42} (1.0)  {3 9 14 16 21 26 31 38 42} (1.0)  {3 10 14 16 21 28 31 37 42} (1.0)  {1 10 13 17 25 28 31 39 42} (1.0)  {2 10 13 16 22 27 32 38 42} (1.0) |
|  |
|  |
|  |

* Mảng PN:

WP WN

Att 1: 2.0 4.0

Att 2: 2.0 2.0

Att 3: 2.0 3.0

Att 4: 0.0 0.0

Att 5: 0.0 0.0

Att 6: 0.0 0.0

Att 7: 0.0 1.0

Att 8: 5.0 0.0

Att 9: 1.0 3.0

Att 10: 0.0 5.0

Att 11: 1.0 0.0

Att 12: 0.0 1.0

Att 13: 2.0 6.0

Att 14: 3.0 2.0

Att 15: 0.0 0.0

Att 16: 4.0 4.0

Att 17: 2.0 4.0

Att 18: 0.0 1.0

Att 19: 0.0 0.0

Att 20: 0.0 0.0

Att 21: 6.0 6.0

Att 22: 0.0 1.0

Att 23: 0.0 0.0

Att 24: 0.0 1.0

Att 25: 0.0 1.0

Att 26: 0.0 1.0

Att 27: 2.0 2.0

Att 28: 4.0 5.0

Att 29: 0.0 1.0

Att 30: 0.0 0.0

Att 31: 5.0 5.0

Att 32: 0.0 3.0

Att 33: 1.0 0.0

Att 34: 0.0 0.0

Att 35: 0.0 1.0

Att 36: 5.0 2.0

Att 37: 0.0 2.0

Att 38: 0.0 4.0

Att 39: 1.0 1.0

Att 40: 0.0 0.0

* Theo bảng P ta có tổng trọng số của P là 6.0 và tổng trọng số ngưỡng (TWT) được tính theo công thức sau với TOTAL\_WEIGHT\_THRESHOLD là 0.05

TWT = totalWeight \* TOTAL\_WEIGHT\_THRESHOLD

= 6.0 \* 0.05 = 0.3

**Bước 3:**

while (total weight P > minimum total Weight threshold)

A' 🡨 A, N' 🡨 N, P' 🡨 P, PN' 🡨 PN

if no attributes exist with weightings that can

produce a gain above minimum: break

* Tập P sau đó được xử lý, cho lớp đã cho đến khi tổng trọng số của nó giảm xuống dưới ngưỡng (0.3). Đầu tiên chúng ta tạo các bản sao của P, N, A và PN là P', N', A' và PN'.
* Mảng P’

Trọng số

{1 8 13 17 21 27 31 36 41} (1.0)

{1 8 13 17 21 28 31 36 41} (1.0)

{2 8 14 16 21 27 31 36 41} (1.0)

{3 8 11 16 21 28 31 36 41} (1.0)

{2 8 14 16 21 28 31 36 41} (1.0)

{3 9 14 16 21 28 33 39 41} (1.0)

* Mảng N’

|  |
| --- |
| Trọng số  {2 9 13 17 21 28 32 38 42} (1.0)  {3 10 13 16 21 27 32 36 42} (1.0)  {1 9 12 17 21 29 35 37 42} (1.0)  {1 7 13 17 21 28 31 36 42} (1.0)  {1 10 13 18 24 28 31 38 42} (1.0)  {3 9 14 16 21 26 31 38 42} (1.0)  {3 10 14 16 21 28 31 37 42} (1.0)  {1 10 13 17 25 28 31 39 42} (1.0)  {2 10 13 16 22 27 32 38 42} (1.0) |
|  |
|  |
|  |

* A’ 🡨 A //Chứa tập các thuộc tính từ 1 - 40
* PN’ = PN //Mảng PN ở trên

Bước 4: Thực hiện phương thức cparGaneration

Với mỗi thuộc tính (1,2,3,…,40) trong mảng A, tính gain cho mỗi thuộc tính và nó sẽ được thêm vào tập luật (mà hiện đang rỗng) và đặt các kết quả đạt được trong mảng thuộc tính

for each a in A

if (a not in antecedent1) calculated gain using

information in PN array and add to attribute array

// An attribute is not in the antecedent if the appropriate cell in the attributes array has the value 0.

* Ví dụ Gain của thuộc tính 1 (Gain1 )

Gain1  = WP1’log(WP1’/WP1’ + PN’1) – log(WP1/WP1 + PN1)

= WP1’log(WP1’/ WP1’ + (WP1 + WN’1)) – log(WP1/WP1 + (WP1 + WN1))

= 2 \* log(2/2 + (2 + 4)) – log(2/2 + 6) = -0.36

* Ta có kết quả như sau đưa vào mảng A

Atributes: Gain

Col 1: 0.0 -0.36

Col 2: 0.0 0.45

Col 3: 0.0 0.0

Col 4: 0.0 0.0

Col 5: 0.0 0.0

Col 6: 0.0 0.0

Col 7: 0.0 0.0

Col 8: 0.0 4.58

Col 9: 0.0 -0.47

Col 10: 0.0 0.0

Col 11: 0.0 0.91

Col 12: 0.0 0.0

Col 13: 0.0 -0.94

Col 14: 0.0 1.21

Col 15: 0.0 0.0

Col 16: 0.0 0.89

Col 17: 0.0 -0.36

Col 18: 0.0 0.0

Col 19: 0.0 0.0

Col 20: 0.0 0.0

Col 21: 0.0 1.33

Col 22: 0.0 0.0

Col 23: 0.0 0.0

Col 24: 0.0 0.0

Col 25: 0.0 0.0

Col 26: 0.0 0.0

Col 27: 0.0 0.45

Col 28: 0.0 0.42

Col 29: 0.0 0.0

Col 30: 0.0 0.0

Col 31: 0.0 1.12

Col 32: 0.0 0.0

Col 33: 0.0 0.92

Col 34: 0.0 0.0

Col 35: 0.0 0.0

Col 36: 0.0 2.90

Col 37: 0.0 0.0

Col 38: 0.0 0.0

Col 39: 0.0 0.22

Col 40: 0.0 0.0

**Bước 5:** Xác định giá trị gain cao nhất và tính giá trị LGT

* Thuộc tính có gain cao nhất là thuộc tính 8 (gain = 4.58 > MIN\_BEST\_GAIN =0.7). Vì vậy Local Gain Threshold (LGT) được tính theo công thức sau, với GAIN\_SIMILARITY\_RATIO được chọn là 0.6:

LGT = 4.58 \* GAIN\_SIMILARITY\_RATIO

= 4.58 \* 0.6 = 2.75

i = "available" column in A with best gain

if (A[i][0] <= MIN\_BEST\_GAIN) return

loop through attribute array and find attribute a' with best gain

if (best gain <= MIN\_BEST\_GAIN) {

add antecedent->c to rule list

for all records in P reduce weighting by decay factor

and adjust PN array accordingly

return

gainThreshold = bestGain\*GAIN\_SIMILARITY\_RATIO

Bước 6: Xác định các thuộc tính trong A có giá trị gain lớn hơn giá trị LGT thì cập nhật lại P’, N’, A’, NP’ và thêm luật đó vào tập luật

for each a in A

if a available1 and a.gain>gainThreshold

tempP' 🡨 P', tempN' 🡨 N', tempA' 🡨 A', tempNP'🡨NP'

add a' to antecedent

remove examples from N' and P' that do not contain

antecedent and adjust NP array accordingly

if (N' == {})

add antecedent->c to rule list

for all roecords in P reduce wieighting by decay factor

and adjust NP array accordingly

else prmGeneration(antecedent,c)

P' 🡨 tempP, N' 🡨 tempN', A' 🡨 tempA', NP' 🡨 tempNP'

end loop

* Dựa vào bảng thuộc tính A ta thấy thuộc tính 8, 36 có gain lớn hơn LGT
* Đầu tiên, ta sẽ thực hiện với thuộc tính 8.
* Tạo bản sao của P', N', A' và PN'; và tiếp tục. với những bản sao này, như sau:
* Thêm thuộc tính 8 vào tập luật điều kiện 8 🡪 41
* Điều chỉnh bản sao P' và N' sao cho tất cả các ví dụ không chứa thuộc tính 8 được loại bỏ và bản sao của mảng PN được điều chỉnh cho phù hợp.
* Mảng P sau điều chỉnh

{1 8 13 17 21 27 31 36 41}

{1 8 13 17 21 28 31 36 41}

{2 8 14 16 21 27 31 36 41}

{3 8 11 16 21 28 31 36 41}

{2 8 14 16 21 28 31 36 41}

* Mảng N sau điều chỉnh = null
* Mảng PN sau điều chỉnh

WP WN

Att 1: 2.0 0.0

Att 2: 2.0 0.0

Att 3: 1.0 0.0

Att 4: 0.0 0.0

Att 5: 0.0 0.0

Att 6: 0.0 0.0

Att 7: 0.0 0.0

Att 8: 5.0 0.0

Att 9: 0.0 0.0

Att 10: 0.0 0.0

Att 11: 1.0 0.0

Att 12: 0.0 0.0

Att 13: 2.0 0.0

Att 14: 2.0 0.0

Att 15: 0.0 0.0

Att 16: 3.0 0.0

Att 17: 2.0 0.0

Att 18: 0.0 0.0

Att 19: 0.0 0.0

Att 20: 0.0 0.0

Att 21: 5.0 0.0

Att 22: 0.0 0.0

Att 23: 0.0 0.0

Att 24: 0.0 0.0

Att 25: 0.0 0.0

Att 26: 0.0 0.0

Att 27: 2.0 0.0

Att 28: 3.0 0.0

Att 29: 0.0 0.0

Att 30: 0.0 0.0

Att 31: 5.0 0.0

Att 32: 0.0 0.0

Att 33: 0.0 0.0

Att 34: 0.0 0.0

Att 35: 0.0 0.0

Att 36: 5.0 0.0

Att 37: 0.0 0.0

Att 38: 0.0 0.0

Att 39: 0.0 0.0

Att 40: 0.0 0.0

* N' bây giờ rỗng, do đó luật 8 -> 41 được chèn vào danh sách luật. Ta điều chỉnh trọng số (weighting) trong P theo hệ số phân rã (decay factor)
* Mảng P tương ứng.

Trọng số sau điều chỉnh

{1 8 13 17 21 27 31 36 41} (0.33)

{1 8 13 17 21 28 31 36 41} (0.33)

{2 8 14 16 21 27 31 36 41} (0.33)

{3 8 11 16 21 28 31 36 41} (0.33)

{2 8 14 16 21 28 31 36 41} (0.33)

{3 9 14 16 21 28 33 39 41} (1.0)

newWP = oldWP \* 1/3

* Quay trở lại danh sách thuộc tính và tiếp tục xử lý tìm kiếm thuộc tính với gain trên mức tối thiểu là 2.75 và tìm thuộc tính 36 (gain = 2.9). Thêm thuộc tính 36 vào tập luật điều kiện 36 -> 41 và tiếp tục xử lý với các bản sao của N', P', A' và NP' như thuộc tính 8.
* Các bản sao của N' và P' được giảm và bản sao của NP' được điều chỉnh để cung cấp cho:
* Bản sao P’

Trọng số

{1 8 13 17 21 27 31 36 41} (1.0)

{1 8 13 17 21 28 31 36 41} (1.0)

{2 8 14 16 21 27 31 36 41} (1.0)

{3 8 11 16 21 28 31 36 41} (1.0)

{2 8 14 16 21 28 31 36 41} (1.0)

* Bản sao N’ = null
* Bản sao mảng NP’

WP’ WN

Att 1: 2.0 0.0

Att 2: 2.0 0.0

Att 3: 1.0 0.0

Att 4: 0.0 0.0

Att 5: 0.0 0.0

Att 6: 0.0 0.0

Att 7: 0.0 0.0

Att 8: 5.0 0.0

Att 9: 0.0 0.0

Att 10: 0.0 0.0

Att 11: 1.0 0.0

Att 12: 0.0 0.0

Att 13: 2.0 0.0

Att 14: 2.0 0.0

Att 15: 0.0 0.0

Att 16: 3.0 0.0

Att 17: 2.0 0.0

Att 18: 0.0 0.0

Att 19: 0.0 0.0

Att 20: 0.0 0.0

Att 21: 5.0 0.0

Att 22: 0.0 0.0

Att 23: 0.0 0.0

Att 24: 0.0 0.0

Att 25: 0.0 0.0

Att 26: 0.0 0.0

Att 27: 2.0 0.0

Att 28: 3.0 0.0

Att 29: 0.0 0.0

Att 30: 0.0 0.0

Att 31: 5.0 0.0

Att 32: 0.0 0.0

Att 33: 0.0 0.0

Att 34: 0.0 0.0

Att 35: 0.0 0.0

Att 36: 5.0 0.0

Att 37: 0.0 0.0

Att 38: 0.0 0.0

Att 39: 0.0 0.0

Att 40: 0.0 0.0

* Bản sao của N' bây giờ rỗng, do đó quy tắc 36 -> 41 được chèn vào danh sách luật và P được điều chỉnh cho phù hợp:

newWP = oldWP \* 1/3 = 0.33 \* 1/3 = 0.11

* Mảng P được điều chỉnh tiếp tục

Trọng số sau điều chỉnh

{1 8 13 17 21 27 31 36 41} (0.11)

{1 8 13 17 21 28 31 36 41} (0.11)

{2 8 14 16 21 27 31 36 41} (0.11)

{3 8 11 16 21 28 31 36 41} (0.11)

{2 8 14 16 21 28 31 36 41} (0.11)

{3 9 14 16 21 28 33 39 41} (1.0)

* P bây giờ đã có tất cả các trọng số đáp ứng các luật 8 -> 41 và 36-> 41 giảm bởi hệ số phân rã nhất định là 1/3.
* Mảng NP bây giờ là:

Att 1: 0.22 4.0

Att 2: 0.22 2.0

Att 3: 1.11 3.0

Att 4: 0.0 0.0

Att 5: 0.0 0.0

Att 6: 0.0 0.0

Att 7: 0.0 1.0

Att 8: 0.55 0.0

Att 9: 1.0 3.0

Att 10: 0.0 5.0

Att 11: 0.11 0.0

Att 12: 0.0 1.0

Att 13: 0.22 6.0

Att 14: 1.22 2.0

Att 15: 0.0 0.0

Att 16: 1.33 4.0

Att 17: 0.22 4.0

Att 18: 0.0 1.0

Att 19: 0.0 0.0

Att 20: 0.0 0.0

Att 21: 1.55 6.0

Att 22: 0.0 1.0

Att 23: 0.0 0.0

Att 24: 0.0 1.0

Att 25: 0.0 1.0

Att 26: 0.0 1.0

Att 27: 0.22 2.0

Att 28: 1.33 5.0

Att 29: 0.0 1.0

Att 30: 0.0 0.0

Att 31: 0.55 5.0

Att 32: 0.0 3.0

Att 33: 1.0 0.0

Att 34: 0.0 0.0

Att 35: 0.0 1.0

Att 36: 0.55 2.0

Att 37: 0.0 2.0

Att 38: 0.0 4.0

Att 39: 1.0 1.0

Att 40: 0.0 0.0

* Ngưỡng tổng trọng số (TWT) cho P bây giờ là 1.55, trên ngưỡng tối thiểu là 0.3, vì vậy lặp lại quá trình với các tập mẫu được sửa đổi P' (được sao chép từ phiên bản sửa đổi của P) và N' (được sao chép từ bản gốc N):

Mảng P’ được sao chép từ mảng sửa đổi của P:

Mảng N’ được sao chép từ mảng N gốc

Trọng số

{2 9 13 17 21 28 32 38 42} (1.0)

{3 10 13 16 21 27 32 36 42} (1.0)

{1 9 12 17 21 29 35 37 42} (1.0)

{1 7 13 17 21 28 31 36 42} (1.0)

{1 10 13 18 24 28 31 38 42} (1.0)

{3 9 14 16 21 26 31 38 42} (1.0)

{3 10 14 16 21 28 31 37 42} (1.0)

{1 10 13 17 25 28 31 39 42} (1.0)

{2 10 13 16 22 27 32 38 42} (1.0)

Trọng số

{1 8 13 17 21 27 31 36 41} (0.11)

{1 8 13 17 21 28 31 36 41} (0.11)

{2 8 14 16 21 27 31 36 41} (0.11)

{3 8 11 16 21 28 31 36 41} (0.11)

{2 8 14 16 21 28 31 36 41} (0.11)

{3 9 14 16 21 28 33 39 41} (1.0)

* Ta cũng tạo ra được 1 mảng NP’ mới có trọng số tích cực từ mảng NP’ và trọng số tiêu cực từ mảng NP trước đó

Mảng NP’

WP’ WN’

Att 1: 0.22 4.0

Att 2: 0.22 2.0

Att 3: 1.11 3.0

Att 4: 0.0 0.0

Att 5: 0.0 0.0

Att 6: 0.0 0.0

Att 7: 0.0 1.0

Att 8: 0.56 0.0

Att 9: 1.0 3.0

Att 10: 0.0 5.0

Att 11: 0.11 0.0

Att 12: 0.0 1.0

Att 13: 0.22 6.0

Att 14: 1.22 2.0

Att 15: 0.0 0.0

Att 16: 1.33 4.0

Att 17: 0.22 4.0

Att 18: 0.0 1.0

Att 19: 0.0 0.0

Att 20: 0.0 0.0

Att 21: 1.56 6.0

Att 22: 0.0 1.0

Att 23: 0.0 0.0

Att 24: 0.0 1.0

Att 25: 0.0 1.0

Att 26: 0.0 1.0

Att 27: 0.22 2.0

Att 28: 1.33 5.0

Att 29: 0.0 1.0

Att 30: 0.0 0.0

Att 31: 0.56 5.0

Att 32: 0.0 3.0

Att 33: 1.0 0.0

Att 34: 0.0 0.0

Att 35: 0.0 1.0

Att 36: 0.56 2.0

Att 37: 0.0 2.0

Att 38: 0.0 4.0

Att 39: 1.0 1.0

Att 40: 0.0 0.0

* Tạo một mảng thuộc tính trống A' (bằng cách sao chép A). Như trước đây, tính toán tất cả gain cho mỗi thuộc tính (và lưu trữ trong mảng thuộc tính). Thuộc tính có gain tốt nhất bây giờ là thuộc tính 33 (gain = 0.92), tương tự ta có Local Gain Threshold (LGT) sẽ được tính như sau:

LGT = 0.92\*GAIN\_SIMILARITY\_RATIO

= 0.92\*0.6 = 0.55

* LGT ban đầu được chỉ định là 0.7 > 0.55, vì vậy 0.7 sẽ được sử dụng làm vùng tối thiểu. Do đó chỉ có một thuộc tính trên ngưỡng này (33). Điều này được chèn vào luật điều kiện để cung cấp cho 33 -> 41 và các bản sao của các bộ mẫu P' và N' và bản sao của mảng NP' được điều chỉnh cho phù hợp. N' hiện đang trống vì vậy quy tắc 33 -> 41 được chèn vào danh sách quy tắc.
* Bây giờ chúng ta lặp lại. Tổng trọng số cho P bây giờ là 0,89, trên ngưỡng tổng trọng số là 0,3.
* Khi kết thúc quá trình, chúng ta sẽ có danh sách luật sau. Lưu ý rằng các quy tắc được chèn vào danh sách luật, vì chúng được sắp xếp theo tính toán Laplace tính chính xác của chúng.

(1) {8} -> {41} 0.86%

(2) {10} -> {42} 0.86%

(3) {38} -> {42} 0.83%

(4) {32} -> {42} 0.8%

(5) {36} -> {41} 0.67%

(6) {33} -> {41} 0.67%

TÀI LIỆU THAM KHẢO :

[1] : THE LUCS-KDD IMPLEMENTATIONS OF CPAR (CLASSIFICATION BASED ON PREDICTIVE ASSOCIATION RULES) – Frans Coenen – 16 February 2004

[2]: X.Yin and J.Han, “CPAR: Classification Based on Predictive Association Rules”, Proceedings of SIAM International conference on Data Mining 2003, pp.331-335