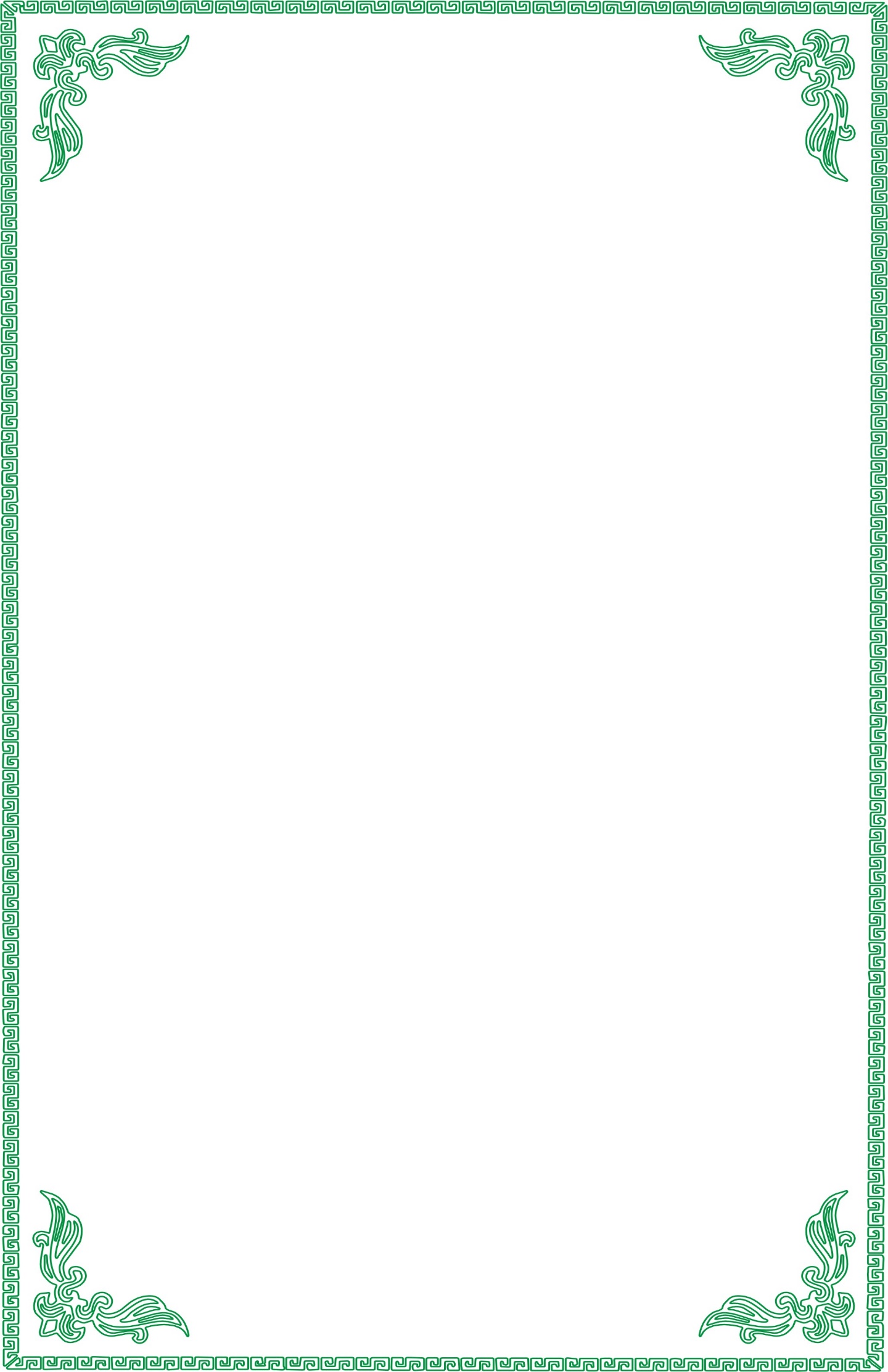
****

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

**MÔN: MÁY HỌC**



**CASE STUDY #1**

**CLASSIFICATION – TITANIC DATASET**

**Giảng viên: Huỳnh Thị Thanh Thương**

**Sinh viên thực hiện**:

17520324 – Nguyễn Thành Danh

17521244 – Hồ Sỹ Tuyến

17520828 – Phan Nguyên

Nội dung Case Study #1

[I. Define a learning problem (Phát biểu bài toán) 4](#_Toc4949484)

[1. Task (Mục đích) 4](#_Toc4949485)

[2. Input (Đầu vào) 4](#_Toc4949486)

[3. Output (Đầu ra) 4](#_Toc4949487)

[4. Evaluation Method (Cách đánh giá mô hình) 4](#_Toc4949488)

[II. Build a model (Xây dựng mô hình từ tập huấn luyện) 4](#_Toc4949489)

[1. Chọn phương pháp demo: 4](#_Toc4949490)

[2. Trình bày thuật toán Naive Bayes dưới dạng mã giả: 4](#_Toc4949491)

[3. Minh họa thuật toán: 5](#_Toc4949492)

[III. Performance Evaluation (Thực nghiệm đánh giá) 7](#_Toc4949493)

[1. Tổ chức Dataset: 7](#_Toc4949494)

[2. Cách đánh giá: 7](#_Toc4949495)

[3. Kết quả đánh giá của từng phương pháp: 8](#_Toc4949496)

[i. Sử dụng phương pháp Naive Bayes – code thủ công: 8](#_Toc4949497)

[ii. Sử dụng phương pháp Naive Bayes – code bằng hàm: 9](#_Toc4949498)

[iii. Sử dụng phương pháp K-Nearest Neighbors – code thủ công: 9](#_Toc4949499)

[iv. Sử dụng phương pháp K-Nearest Neighbors – code bằng hàm: 10](#_Toc4949500)

[v. Sử dụng phương pháp Decision Tree – code bằng hàm: 10](#_Toc4949501)

[4. Nhận xét: 10](#_Toc4949502)

[5. Ưu điểm – Hạn chế của các mô hình học: 10](#_Toc4949503)

[IV. Implementation (Lập trình cài đặt) 11](#_Toc4949504)

[1. Các thư viện đã sử dụng: 11](#_Toc4949505)

[2. Hướng dẫn cài đặt tools và giải thích các hàm đã sử dụng: 12](#_Toc4949506)

[i. K-Nearest Neighbors:(6)(7) 12](#_Toc4949507)

[ii. Naive Bayes:(8)(9) 14](#_Toc4949508)

[iii. Decision Tree:(10)(11) 15](#_Toc4949509)

[3. Source Code: 16](#_Toc4949510)

[i. Pre\_Processing\_Data: *(Tiền xử lý dataset)* 16](#_Toc4949511)

[ii. KNN\_sk.py: *(K-Nearest Neighbors dùng thư viện Sklearn)* 18](#_Toc4949512)

[iii. KNN\_np.py: *(K-Nearest Neighbors code thủ công)* 19](#_Toc4949513)

[iv. Naive\_Bayes\_sk.py: *(Naive Bayes dùng thư viện Sklearn)* 21](#_Toc4949514)

[v. Naive\_Bayes\_np.py: *(Naive Bayes code thủ công)* 23](#_Toc4949515)

[vi. Decision\_Tree\_sk.py: *(Decision Tree dùng thư viện Sklearn)* 28](#_Toc4949516)

[4. Demo kết quả chụp màn hình: 30](#_Toc4949517)

[V. References (Tài liệu tham khảo) 31](#_Toc4949518)

# Define a learning problem (Phát biểu bài toán)

## Task (Mục đích)

* Dựa trên dữ liệu có được từ thảm họa Titanic, bằng các phương pháp học (KNN, Naive Bayes, Decision Tree), khi nhận dữ liệu đầu vào là một số thông tin cơ bản của 1 người sẽ đưa ra được câu trả lời là người đó đã sống sót hay không.

## Input (Đầu vào)

* Tập dữ liệu Titanic Dataset gồm thông tin của 2201 người có mặt trên tàu Titanic xảy ra tai nạn (gồm social class, age, sex, survived or not) được chia thành 2 tập, trainingset : testset với tỉ lệ 2 : 1.

## Output (Đầu ra)

* Dựa trên 3 thuộc tính: social class, age, sex đưa ra được quyết định người đó đã sống sót hay không trên chuyến tàu này. (isAlive/not)

## Evaluation Method (Cách đánh giá mô hình)

* Đánh giá bằng Precision (độ chính xác) và Recall (độ phủ).
* Từ đó tính ra được F1-Score là trung bình điều hòa của 2 tiêu chí trên.
* Cụ thể ở phần [IV Performance Evaluation](#_Performance_Evaluation_(Thực).

# Build a model (Xây dựng mô hình từ tập huấn luyện)

## Chọn phương pháp demo:

Sử dụng phương pháp học Naive Bayes để học 1 mô hình từ tập dữ liệu huấn luyện (2/3 của dataset gồm 1466 inputs)

## Trình bày thuật toán Naive Bayes dưới dạng mã giả:

Thuật toán Naive Bayes gồm 2 giai đoạn: Huấn luyện và phân lớp.

* + - Giai đoạn 1: Huấn luyện.
      * *Input*: Trainingset gồm 1466 vectors chứa các đặc trưng và nhãn cho từng người trên chuyến tàu Titanic. Các đặc trưng gồm: class, age, sex; Nhãn: isAlive.
      * *Output*: Các giá trị xác suất của từng biến cố của mỗi đặc trưng ứng với từng nhãn.

*B1*: Khởi tạo các list chứa các bộ dữ liệu X, y cho trainingset và testset.

*B2*: Duyệt qua các bộ dữ liệu trong trainingset để tính được xác suất của từng biến cố của mỗi đặc trưng tương ứng với nhãn (yes/no) isAlive.

* + - Giai đoạn 2: Phân lớp.
      * *Input*:
        + Các giá trị xác suất đã được tính ra từ giai đoạn 1.
        + Testset gồm 735 bộ các đặc trưng cần được gắn nhãn, và các nhãn sẵn có.
      * *Output*: Nhãn của từng bộ các đặc trưng trong testset.

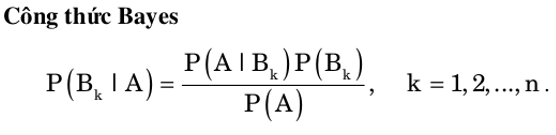
*B3*: Dựa vào xác suất đã tính trên từng đặc trưng, tính ra xác suất của từng nhãn có thể gán cho các Xi của testset, nếu xác suất yes lớn xác suất no thì gán nhãn yes cho Yi của testset đó và ngược lại.

## Minh họa thuật toán:

* + Tập dataset được xáo trộn ngẫu nhiên để phương pháp học hiệu quả hơn, lấy ra 2/3 (1466 bộ) làm thành tập trainingset, 1/3 (735 bộ) còn lại làm thành tập testset.
  + Bảng xác suất sau được lập trên số liệu của tập trainingset

(cụ thể tại file excel - [[CASE\_STUDY\_1]\_REPORT\_Titanic\_Dataset](%5bCASE_STUDY_1%5d_REPORT_Titanic_Dataset%20(17520324_17520828_17521244).xlsx)).

* + Dựa và công thức để tính ra được các giá trị xác xuất trong bảng sau:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Probability Table** | | | | | | | | | | | | | |
| **Class** | | | | **Age** | | | | **Sex** | | | | **Alive** | |
|  | **Count** | **Y** | **N** |  | **Count** | **Y** | **N** |  | **Count** | **Y** | **N** | **Y** | **N** |
| **1st** | 228 | 143 | 85 | **Adult** | 1386 | 434 | 952 | **Male** | 1152 | 239 | 913 | 475 | 991 |
| **2nd** | 189 | 81 | 108 | **Child** | 80 | 41 | 39 | **Female** | 314 | 236 | 78 |
| **3rd** | 474 | 115 | 359 |  |  |  |  |  |  |  |  |
| **Crew** | 575 | 136 | 439 |  |  |  |  |  |  |  |  |
| **1st** |  | 0.301 | 0.086 | **Adult** |  | 0.914 | 0.961 | **Male** |  | 0.5 | 0.92 | 0.32 | 0.68 |
| **2nd** |  | 0.171 | 0.109 | **Child** |  | 0.086 | 0.039 | **Female** |  | 0.5 | 0.08 |
| **3rd** |  | 0.242 | 0.362 |  |  |  |  |  |  |  |  |
| **Crew** |  | 0.286 | 0.443 |  |  |  |  |  |  |  |  |

* + Thử tính cho 2 trường hợp đầu tiên của testset:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Class | Age | Sex | Alive |
| 1467 | crew | adult | male | yes |
| 1468 | 1st | adult | female | yes |

* **STT 1467:**

**P(Alive=Y) = = = 0.1336 P(Alive=N) = = = 0.8664**

***Kết quả 1:*** *Với tỉ lệ 0.13 Yes và 0.87 No, TH1 của testset qua Naive Bayes cho ra kết quả No, sai so với kết quả từ testset.*

* **STT 1468:**

**P(Alive=Y) = = = 0.9167**

**P(Alive=N) = = = 0.0833**

***Kết quả 2:*** *Với tỉ lệ 0.92 Yes và 0.08 No, TH2 của testset qua Naive Bayes cho ra kết quả Yes, đúng so với kết quả từ testset.*

Tất cả các trường hợp còn lại của tập testset đã được tính trong file excel, kết quả cuối cùng thu được: 562 kết quả đúng và 173 kết quả sai so với tập testset.

# Performance Evaluation (Thực nghiệm đánh giá)

## Tổ chức Dataset:

* Dataset ([dataset.data](Dataset.data)) gồm 2201 bộ dữ liệu (đặc trưng + nhãn) được trộn ngẫu nhiên theo hàm random.shuffle() trở thành file [My\_Data.txt](Source_Code_Titanic_Dataset/My_Data.txt) gồm 2201 bộ dữ liệu được mã hóa theo 1 quy tắc riêng để dễ thao tác. (quy tắc xem bảng Convention trong file excel)
* Trainingset: 1466 bộ dữ liệu đầu tiên của My\_Data.txt
* Testset: 735 bộ dữ liệu còn lại của My\_Data.txt

*(Tỉ lệ Trainingset : Testset = 2 : 1)*

## Cách đánh giá:

* Đánh giá bằng **Precision** (xem xét trong testset có bao nhiêu dữ liệu được mô hình dự đoán đúng) và **Recall** (xem xét trong số các dữ liệu được mô hình dự đoán đúng có bao nhiêu dữ liệu là đúng). Theo công thức[(1)](https://machinelearningcoban.com/2017/08/31/evaluation/#-precision-va-recall):

Trong đó:

* Lớp positive là lớp có giá trị y = 1.
* TP (True Positive): tổng số các mẫu mà mô hình dự đoán là positive và thực sự có nhãn positive y = 1.
* FP (False Positive): là tổng số các mẫu mô hình dự đoán là positive nhưng thực sự có nhãn là negative y = 0.
* P là tổng số mẫu positive trong testset.
  + Từ đó tính ra được **F1-Score** là trung bình điều hòa của 2 tiêu chí trên, nó có xu hướng lấy giá trị gần với giá trị nhỏ hơn giữa Precision và Recall đồng thời có giá trị lớn nếu cả Precision và Recall đều lớn.

## Kết quả đánh giá của từng phương pháp:

### Sử dụng phương pháp Naive Bayes – code thủ công:

* Dựa vào các số liệu xác suất tính được từ trainingset, mô hình dự đoán trên tập testing cho kết quả (cụ thể ở file excel) như sau:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RESULT TABLE** | | | | | | |
| **Num** | **p(x)yes** | **isAlive=yes** | **p(x)no** | **isAlive=no** | **conclu** | **compar** |
| 1467 | 0.043 | **0.14** | 0.265 | **0.86** | **no** | 0 |
| 1468 | 0.044 | **0.917** | 0.004 | **0.083** | **yes** | 1 |
| 1469 | 0.036 | **0.655** | 0.019 | **0.345** | **yes** | 0 |
| 1470 | 0.043 | **0.14** | 0.265 | **0.86** | **no** | 1 |
| 1471 | 0.036 | **0.142** | 0.217 | **0.858** | **no** | 0 |
| ... |  |  |  |  |  |  |
| 2200 | 0.036 | **0.142** | 0.217 | **0.858** | **no** | 1 |
| 2201 | 0.036 | **0.142** | 0.217 | **0.858** | **no** | 0 |

Trong đó:

* Num: số thứ tự các bộ trong testset.
* p(x)yes: xác suất các đặc trưng dẫn đến nhãn là yes.
* isAlive=yes: xác suất người đó sống.
* p(x)no: xác suất các đặc trưng dẫn đến nhãn là no.
* isAlive=no: xác suất người đó chết.
* conclu: nhãn của bộ dữ liệu đó được mô hình dự đoán .
* compar: so sánh với nhãn y của bộ dữ liệu đó trong testset, nếu giống  
  (= 1), nếu không giống (= 0).
* Theo công thức, ta tính được Precision, Recall và F1-Score:

Trong đó:

* Lớp positive là lớp có giá trị isAlive = yes.
* TP (True Positive): tổng số các mẫu mà mô hình dự đoán là positive và thực sự có nhãn positive isAlive = yes.
* FP (False Positive): là tổng số các mẫu mô hình dự đoán là positive nhưng thực sự có nhãn là negative isAlive = no.
* P là tổng số mẫu có nhãn là yes trong testset.

*Áp dụng công thức như trên, ta tính cho các phương pháp khác, để ngắn gọn và dễ so sánh, dưới đây xin phép chỉ trình bày kết quả dưới dạng bảng: (chi tiết được thể hiện ở file excel – sheet Results\_Comparision).*

### Sử dụng phương pháp Naive Bayes – code bằng hàm:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | | | |
| **TP** | | **FP** | |
| **114** | | **57** | |
| **0.666666667** | | | |
|  | |  | |
| **Recall** | | | |
| **TP** | | **P** | |
| **114** | | **236** | |
| **0.483050847** | | | |

|  |
| --- |
| **F1-Score** |
| **0.56019656** |

### Sử dụng phương pháp K-Nearest Neighbors – code thủ công:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | | | |
| **TP** | | **FP** | |
| **112** | | **98** | |
| **0.533333333** | | | |
|  | |  | |
| **Recall** | | | |
| **TP** | | **P** | |
| **112** | | **236** | |
| **0.474576271** | | | |

|  |
| --- |
| **F1-Score** |
| **0.502242152** |

### Sử dụng phương pháp K-Nearest Neighbors – code bằng hàm:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | | | |
| **TP** | | **FP** | |
| **119** | | **161** | |
| **0.425** | | | |
|  | |  | |
| **Recall** | | | |
| **TP** | | **P** | |
| **119** | | **236** | |
| **0.504237288** | | | |

|  |
| --- |
| **F1-Score** |
| **0.46124031** |

### Sử dụng phương pháp Decision Tree – code bằng hàm:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precision** | | | |
| **TP** | | **FP** | |
| **80** | | **7** | |
| **0.91954023** | | | |
|  | |  | |
| **Recall** | | | |
| **TP** | | **P** | |
| **80** | | **236** | |
| **0.338983051** | | | |

|  |
| --- |
| **F1-Score** |
| **0.495356037** |

## Nhận xét:

* Code bằng hàm và code thủ công cho ra kết quả có một số khác biệt, lí giải như sau:
  + Phương pháp Naive Bayes: Sai khác do quá trình làm tròn số (không đáng kể).
  + Phương pháp K-Nearest Neighbors: Sai khác do cách xác định khoảng cách giữa các điểm dữ liệu của hàm trong thư viện và code thủ công khác nhau.
  + Phương pháp Decision Tree: chỉ dùng hàm, không code thủ công.
* Tổng kết lại kết quả F1-Score của các phương pháp: (lấy kết quả sử dụng hàm trong thư viện để so sánh)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **K-Nearest Neighbors** | **Naive Bayes** | **Decision Tree** |
| **F1-Score** | **0.4612** | **0.5602** | **0.4953** |

Như vậy, theo cách đánh giá bằng F1-Score, ta suy ra được phương pháp Naive Bayes là mô hình học hiệu quả nhất trong 3 mô hình học được thử nghiệm.

## Ưu điểm – Hạn chế của các mô hình học:

* K-Nearest Neighbors:
  + Ưu:
    - Độ phức tạp tính toán trong quá trình training thấp.
    - Việc dự đoán nhãn của dữ liệu mới đơn giản.
  + Nhược:
    - Khó khăn trong quá trình xác định neighbor gần nhất với điểm đang xét do có thể có nhiều điểm dữ liệu cùng cách đều điểm đang xét, dẫn đến khó khăn trong việc gắn nhãn điểm dữ liệu đó.
    - Tính toán khoảng cách nhiều lần, tốn bộ nhớ đối với các cơ sở dữ liệu nhiều chiều và nhiều điểm dữ liệu, ảnh hưởng hiệu năng của mô hình.
* Naive Bayes:
  + Ưu:
    - Giả định các đặc trưng độc lập nhau thì kết quả sẽ rất tốt do dựa trên xác suất các bộ dữ liệu học được từ trainingset.
  + Nhược:
    - Giả định tất cả các đặc trưng đều độc lập, không tác động lẫn nhau, điều này bất hợp lý trong thực tế.
* Decision Tree:
  + Ưu:
    - Khi tạo thành cây, ta thấy được rõ đặc trưng của data trong trainingset.
    - Có độ ưu tiên giữa các đặc trưng, đặc trưng có tính quyết định cao sẽ là/gần node gốc, các đặc trưng ít quan trọng hơn sẽ ở xa hơn.
  + Nhược:
    - Bỏ qua một số đặc trưng khi đưa ra quyết định.
    - Độ chính xác cao tuy nhiên độ phủ lại kém (rút ra từ thực nghiệm trên dataset).

# Implementation (Lập trình cài đặt)

## Các thư viện đã sử dụng:

* ***Sklearn***: Scikit-learn là một thư viện máy học phần mềm miễn phí cho ngôn ngữ lập trình Python. Nó có các thuật toán phân loại, hồi quy và phân cụm khác nhau bao gồm các support vector machines, random forests, gradient boosting, k-means và DBSCAN, được thiết kế để tương tác với các thư viện khoa học và số của Python NumPy và SciPy.
* ***NumPy***: (viết tắt của Numerical Python) là một thư viện rất cần thiết khi chúng ta xây dựng các ứng dụng Máy học trên Python. Numpy cung cấp các đối tượng và phương thức để làm việc với mảng nhiều chiều và các phép toán đại số tuyến tính. [(5)](https://vimentor.com/vi/lesson/22-gioi-thieu-numpy)Nó là một mô-đun mở rộng mã nguồn mở cho Python, cung cấp các chức năng biên dịch nhanh cho các thao tác toán học và số. Hơn nữa, NumPy làm phong phú ngôn ngữ lập trình Python với các cấu trúc dữ liệu mạnh mẽ để tính toán hiệu quả các mảng và ma trận đa chiều.

## Hướng dẫn cài đặt tools và giải thích các hàm đã sử dụng:

### K-Nearest Neighbors:(6)(7)

#### Thư viện Sklearn:

* **Class KNeighborsClassifier(n\_neighbors):**

**Chức năng:** Khởi tạo phân loại bằng cách thực hiện KNN với tham số n\_neighbors = k.

* **Hàm fit(X,y):**

**Chức năng:** fix model bằng cách sử dụng X làm dữ liệu huấn luyện, y làm giá trị đích (label).

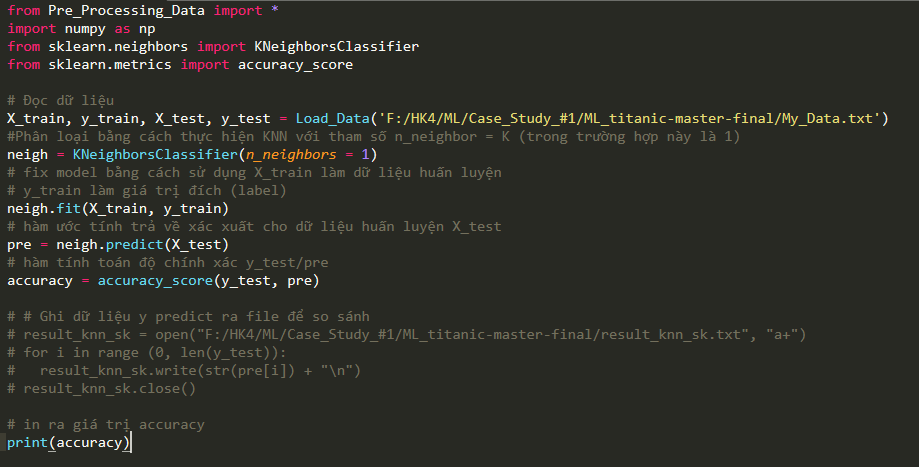
* **Hàm predict(X):**

**Chức năng:** Hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X.

* **Hàm accuracy\_score(y\_true, y\_pre):**

**Chức năng:** Hàm tính toán độ chính xác y\_true và y\_pre.

Hình IV.1



#### Tự định nghĩa hàm:

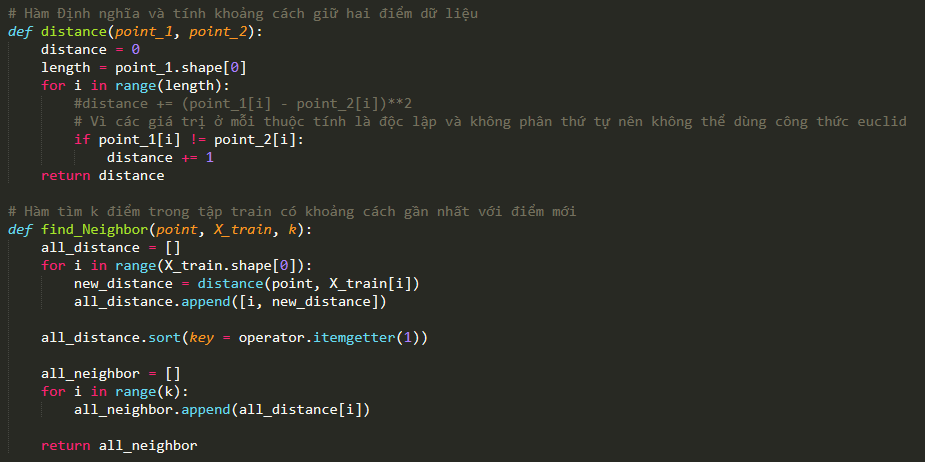
* **Hàm distance(X, Y):**

**Chức năng:** Hàm giúp tính khoảng cách giữa 2 điểm dữ liệu X và Y.

* **Hàm find\_Neighbor(X, X\_train, k):**

**Chức năng**: Hàm giúp tìm k điểm trong tập train có khoảng cách gần nhất với điểm X.

Hình IV.2



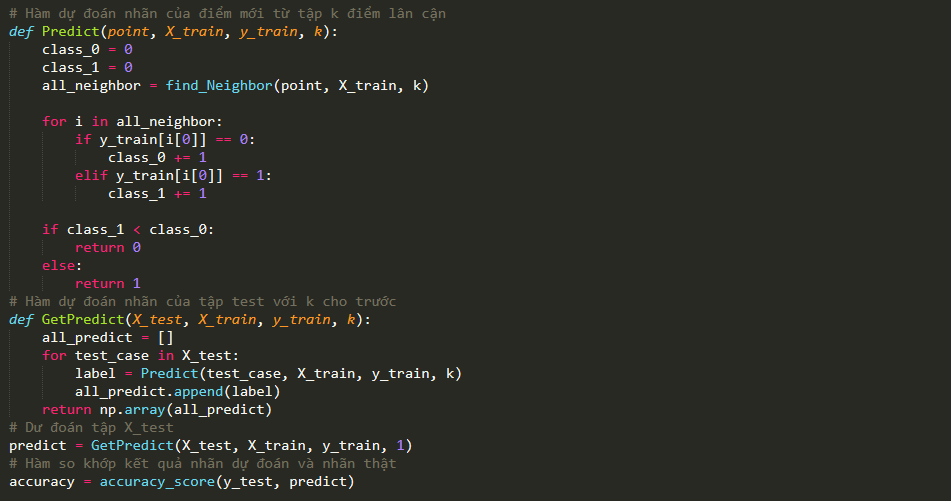
* **Hàm Predict(X, X\_train, y\_train, k):**

**Chức năng:** Hàm dự đoán nhãn của điểm dữ liệu X từ k tập điểm lân cận.

* **Hàm GetPredict(X\_test, X\_train, y\_train, k):**

**Chức năng:** Hàm dự đoán nhãn của tập test với k cho trước.

Hình IV.3



### Naive Bayes:(8)(9)

#### Thư viện Sklearn:

* **Class GausianNB():**

**Chức năng:** Phân loại bằng cách thực hiện Gaussian Naive Bayes.

* **Hàm fit(X, y):**

**Chức năng:** fix model bằng cách sử dụng X làm dữ liệu huấn luyện, y làm giá trị đích (label).

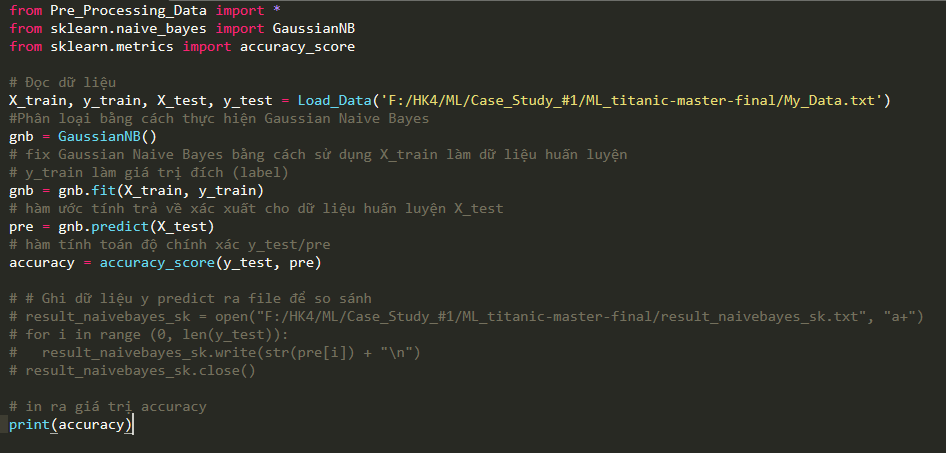
* **Hàm predict(X):**

**Chức năng:** Hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X.

* **Hàm accuracy\_score(y\_true, y\_pre):**

**Chức năng:** Hàm tính toán độ chính xác y\_true và y\_pre.

Hình IV.4



#### Tự định nghĩa hàm:

* **Hàm Probability(X, y):** *(xem tại source code bên dưới)*

**Chức năng:** Hàm tính xác xuất các feature tương ứng với yes và no của label.

* **Hàm Predict(X\_test, y\_test, X\_train, y\_train):** *(xem tại source code bên dưới)*

**Chức năng:** Hàm tính độ chính xác của tập test.

### Decision Tree:(10)(11)

* **Class DecisionTreeClassifier():**

**Chức năng:** Khởi tạo phân loại bằng Decision Tree.

* **Hàm fit(X,y):**

**Chức năng:** fix model bằng cách sử dụng X làm dữ liệu huấnluyện, y làm giá trị đích (label).

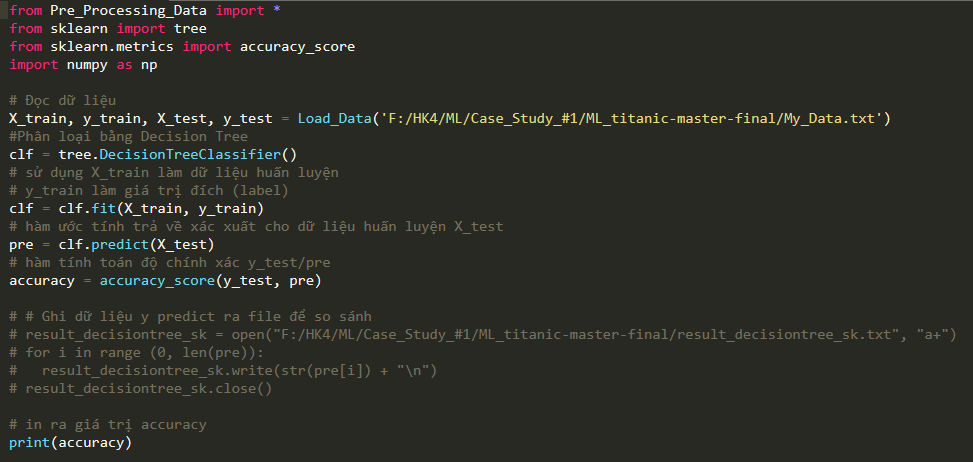
* **Hàm predict(X):**

**Chức năng**: Hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X.

* **Hàm accuracy\_score(y\_true, y\_pre):**

**Chức năng:** Hàm tính toán độ chính xác y\_true và y\_pre.

Hình IV.5



## Source Code:

### Pre\_Processing\_Data: *(Tiền xử lý dataset)*

|  |
| --- |
| import random  import numpy as np  def Pre\_Processing\_Data(path, My\_data):  data = []  with open(path, 'r') as f:  for line in f:  line = line.split('\n')  new = line[0].split(" ")  if '1st' in new:  social\_class = 0  elif '2nd' in new:  social\_class = 1  elif '3rd' in new:  social\_class = 2  elif 'crew' in new:  social\_class = 3  if 'adult' in new:  age = 0  elif 'child' in new:  age = 1  if 'male' in new:  sex = 0  elif 'female' in new:  sex = 1  if 'yes' in new:  isAlive = 1  elif 'no' in new:  isAlive = 0  My\_data.write(str(social\_class) + " " + str(age) + " " + str(sex) + " " + str(isAlive) + "\n")  def Shuffle\_Data(path, My\_data):  data = []  with open(path, 'r') as f:  for line in f:  line = line.split('\n')  line = line[0].split(" ")  data.append([int(line[0]), int(line[1]), int(line[2]), int(line[3])])  random.shuffle(data)  for i in data:  My\_data.write(str(i[0]) + " " + str(i[1]) + " " + str(i[2]) + " " + str(i[3]) + "\n")    def SplitData(My\_data):  data = np.array(My\_data)  X = data[:, 0:3]  y = data[:, 3]  # Chia dữ liệu train và test  a = len(X)//3 # tỉ lệ train : test = 2:1  X\_train = X[0:2\*a]  y\_train = y[0:2\*a]  X\_test = X[2\*a:]  y\_test = y[2\*a:]  return X\_train, y\_train, X\_test, y\_test  def Load\_Data(path):  data = []  with open(path, 'r') as f:  for line in f:  line = line.split('\n')  line = line[0].split(" ")  data.append([int(line[0]), int(line[1]), int(line[2]), int(line[3])])  return SplitData(data) |

### KNN\_sk.py: *(K-Nearest Neighbors dùng thư viện Sklearn)*

|  |
| --- |
| from Pre\_Processing\_Data import \*  from sklearn import tree  from sklearn.metrics import accuracy\_score  import numpy as np  # Đọc dữ liệu  X\_train, y\_train, X\_test, y\_test = Load\_Data('F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/My\_Data.txt')  #Phân loại bằng Decision Tree  clf = tree.DecisionTreeClassifier()  # sử dụng X\_train làm dữ liệu huấn luyện  # y\_train làm giá trị đích (label)  clf = clf.fit(X\_train, y\_train)  # hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X\_test  pre = clf.predict(X\_test)  # hàm tính toán độ chính xác y\_test/pre  accuracy = accuracy\_score(y\_test, pre)  # # Ghi dữ liệu y predict ra file để so sánh  # result\_decisiontree\_sk = open("F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/result\_decisiontree\_sk.txt", "a+")  # for i in range (0, len(pre)):  # result\_decisiontree\_sk.write(str(pre[i]) + "\n")  # result\_decisiontree\_sk.close()  #Tính F1\_Score - Độ đo hiệu quả của mô hình  def F1\_Score(pre, y\_test):  precision = 0 # = TP / (TP + FP)  recall = 0 # = TP / P  TP = 0  FP = 0  P = 0  F1 = 0  for i in range (0, len(y\_test)):  if pre[i] == 1 and y\_test[i] == 1:  TP += 1  elif pre[i] == 1 and y\_test[i] == 0:  FP += 1  precision = TP / (TP + FP)  print("Precision = ", precision)  for i in range (0, len(y\_test)):  if y\_test[i] == 1:  P += 1  recall = TP / P  print("Recall = ", recall)  F1 = 2\*precision\*recall/(precision + recall)  return F1  print("F1\_Score = ", F1\_Score(pre, y\_test))  # in ra giá trị accuracy  print("Accuracy resulted by sklearn = ", accuracy) |

### KNN\_np.py: *(K-Nearest Neighbors code thủ công)*

|  |
| --- |
| from Pre\_Processing\_Data import \*  import numpy as np  import operator  from sklearn.metrics import accuracy\_score  # Đọc dữ liệu  X\_train, y\_train, X\_test, y\_test = Load\_Data('F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/My\_Data.txt')  # Hàm Định nghĩa và tính khoảng cách giữ hai điểm dữ liệu  def distance(point\_1, point\_2):  distance = 0  length = point\_1.shape[0]  for i in range(length):  #distance += (point\_1[i] - point\_2[i])\*\*2  # Vì các giá trị ở mỗi thuộc tính là độc lập và không phân thứ tự nên không thể dùng công thức euclid  if point\_1[i] != point\_2[i]:  distance += 1  return distance  # Hàm tìm k điểm trong tập train có khoảng cách gần nhất với điểm mới  def find\_Neighbor(point, X\_train, k):  all\_distance = []  for i in range(X\_train.shape[0]):  new\_distance = distance(point, X\_train[i])  all\_distance.append([i, new\_distance])    all\_distance.sort(key = operator.itemgetter(1))    all\_neighbor = []  for i in range(k):  all\_neighbor.append(all\_distance[i])  return all\_neighbor  # Hàm dự đoán nhãn của điểm mới từ tập k điểm lân cận  def Predict(point, X\_train, y\_train, k):  class\_0 = 0  class\_1 = 0  all\_neighbor = find\_Neighbor(point, X\_train, k)  for i in all\_neighbor:  if y\_train[i[0]] == 0:  class\_0 += 1  elif y\_train[i[0]] == 1:  class\_1 += 1  if class\_1 < class\_0:  return 0  else:  return 1  # Hàm dự đoán nhãn của tập test với k cho trước  def GetPredict(X\_test, X\_train, y\_train, k):  all\_predict = []  for test\_case in X\_test:  label = Predict(test\_case, X\_train, y\_train, k)  all\_predict.append(label)  return np.array(all\_predict)  # Dư đoán tập X\_test  pre = GetPredict(X\_test, X\_train, y\_train, 1)  # Hàm so khớp kết quả nhãn dự đoán và nhãn thật  accuracy = accuracy\_score(y\_test, pre)  # # Ghi dữ liệu y predict ra file để so sánh  # result\_knn\_np = open("F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/result\_knn\_np.txt", "a+")  # for i in range (0, len(pre)):  # result\_knn\_np.write(str(pre[i]) + "\n")  # result\_knn\_np.close()  #Tính F1\_Score - Độ đo hiệu quả của mô hình  def F1\_Score(pre, y\_test):  precision = 0 # = TP / (TP + FP)  recall = 0 # = TP / P  TP = 0  FP = 0  P = 0  F1 = 0  for i in range (0, len(y\_test)):  if pre[i] == 1 and y\_test[i] == 1:  TP += 1  elif pre[i] == 1 and y\_test[i] == 0:  FP += 1  precision = TP / (TP + FP)  print("Precision = ", precision)  for i in range (0, len(y\_test)):  if y\_test[i] == 1:  P += 1  recall = TP / P  print("Recall = ", recall)  F1 = 2\*precision\*recall/(precision + recall)  return F1  print("F1\_Score = ", F1\_Score(pre, y\_test))  # in ra giá trị accuracy  print("Accuracy resulted by sklearn = ", accuracy) |

### Naive\_Bayes\_sk.py: *(Naive Bayes dùng thư viện Sklearn)*

|  |
| --- |
| from Pre\_Processing\_Data import \*  from sklearn.naive\_bayes import GaussianNB  from sklearn.metrics import accuracy\_score  # Đọc dữ liệu  X\_train, y\_train, X\_test, y\_test = Load\_Data('F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/My\_Data.txt')  #Phân loại bằng cách thực hiện Gaussian Naive Bayes  gnb = GaussianNB()  # fix Gaussian Naive Bayes bằng cách sử dụng X\_train làm dữ liệu huấn luyện  # y\_train làm giá trị đích (label)  gnb = gnb.fit(X\_train, y\_train)  # hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X\_test  pre = gnb.predict(X\_test)  # hàm tính toán độ chính xác y\_test/pre  accuracy = accuracy\_score(y\_test, pre)  # # Ghi dữ liệu y predict ra file để so sánh  # result\_naivebayes\_sk = open("F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/result\_naivebayes\_sk.txt", "a+")  # for i in range (0, len(y\_test)):  # result\_naivebayes\_sk.write(str(pre[i]) + "\n")  # result\_naivebayes\_sk.close()  #Tính F1\_Score - Độ đo hiệu quả của mô hình  def F1\_Score(pre, y\_test):  precision = 0 # = TP / (TP + FP)  recall = 0 # = TP / P  TP = 0  FP = 0  P = 0  F1 = 0  for i in range (0, len(y\_test)):  if pre[i] == 1 and y\_test[i] == 1:  TP += 1  elif pre[i] == 1 and y\_test[i] == 0:  FP += 1  precision = TP / (TP + FP)  print("Precision = ", precision)  for i in range (0, len(y\_test)):  if y\_test[i] == 1:  P += 1  recall = TP / P  print("Recall = ", recall)  F1 = 2\*precision\*recall/(precision + recall)  return F1  print("F1\_Score = ", F1\_Score(pre, y\_test))  # in ra giá trị accuracy  print("Accuracy resulted by sklearn = ", accuracy) |

### Naive\_Bayes\_np.py: *(Naive Bayes code thủ công)*

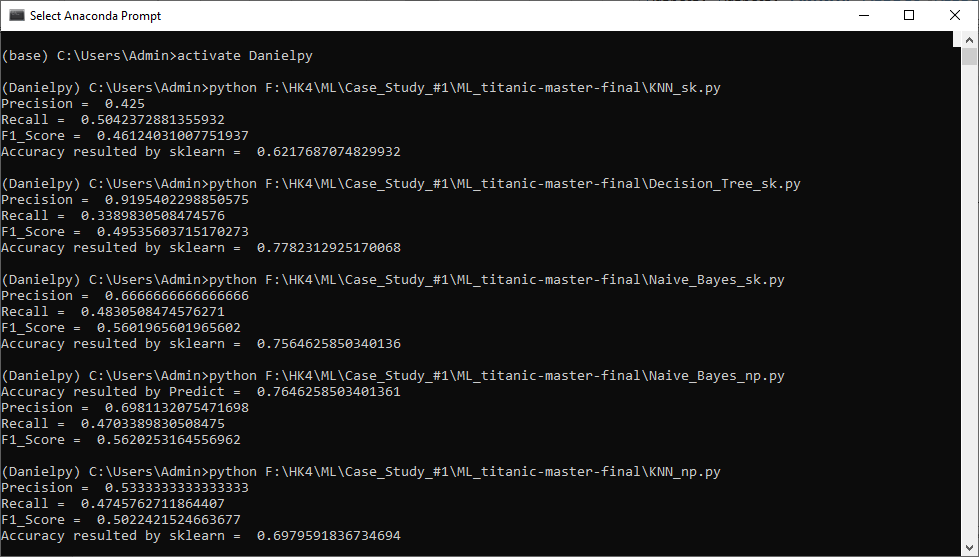
|  |
| --- |
| from Pre\_Processing\_Data import \*  import numpy as np  # Đọc dữ liệu  X\_train, y\_train, X\_test, y\_test = Load\_Data('F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/My\_Data.txt')  # # Ghi dữ liệu y predict ra file để so sánh  # result\_naivebayes\_np = open("F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/result\_naivebayes\_np.txt", "a+")  # hàm tính xác xuất các feature tương ứng với yes và no của label  def Probability(X\_train,y\_train):  m = len(y\_train)  #tổng số trường hợp alive  alive = 0  #tổng số trường hợp died  death = 0  for i in range(m):  if y\_train[i] == 1:  alive += 1  else:  death += 1    live = alive/m  die = death/m  #tỉ lệ giữa sống và chết  R=[live,die]  Class\_1\_alive = 0  Class\_1\_died = 0  Class\_2\_alive = 0  Class\_2\_died = 0  Class\_3\_alive = 0  Class\_3\_died = 0  Class\_4\_alive = 0  Class\_4\_died = 0  Age\_Adult\_alive = 0  Age\_Adult\_died = 0  Age\_Child\_alive = 0  Age\_Child\_died = 0  Male\_alive = 0  Male\_died = 0  Female\_alive = 0  Female\_died = 0  for i in range(m):  if(X\_train[i][0] == 0):    if(y\_train[i] == 1):  Class\_1\_alive+=1  else:  Class\_1\_died+=1  elif(X\_train[i][0] == 1):    if(y\_train[i] == 1):  Class\_2\_alive+=1  else:  Class\_2\_died+=1  elif(X\_train[i][0] == 2):    if(y\_train[i] == 1):  Class\_3\_alive+=1  else:  Class\_3\_died+=1  elif(X\_train[i][0] == 3):    if(y\_train[i] == 1):  Class\_4\_alive+=1  else:  Class\_4\_died+=1  Accuracy\_H1\_alive = Class\_1\_alive/alive  Accuracy\_H2\_alive = Class\_2\_alive/alive  Accuracy\_H3\_alive = Class\_3\_alive/alive  Accuracy\_H4\_alive = Class\_4\_alive/alive  Accuracy\_H1\_death = Class\_1\_died/death  Accuracy\_H2\_death = Class\_2\_died/death  Accuracy\_H3\_death = Class\_3\_died/death  Accuracy\_H4\_death = Class\_4\_died/death  # tỉ lệ sống và chết của loại khách hàng  H=[[Accuracy\_H1\_alive,Accuracy\_H1\_death],  [Accuracy\_H2\_alive,Accuracy\_H2\_death],  [Accuracy\_H3\_alive,Accuracy\_H3\_death],  [Accuracy\_H4\_alive,Accuracy\_H4\_death]]  if X\_train[i][1] == 0:    if(y\_train[i] == 1):  Age\_Adult\_alive+=1  else:  Age\_Adult\_died+=1  elif X\_train[i][1] == 1:    if(y\_train[i] == 1):  Age\_Child\_alive+=1  else:  Age\_Child\_died+=1  Accuracy\_AgeA\_alive = Age\_Adult\_alive/alive  Accuracy\_AgeC\_alive = Age\_Child\_alive/alive  Accuracy\_AgeC\_death = Age\_Child\_died/death  Accuracy\_AgeA\_death = Age\_Adult\_died/death  # tỉ lệ sống và chết của tuổi khách hàng  A=[[Accuracy\_AgeA\_alive,Accuracy\_AgeA\_death],  [Accuracy\_AgeC\_alive,Accuracy\_AgeC\_death]]  if X\_train[i][2] == 0:    if(y\_train[i] == 1):  Male\_alive+=1  else:  Male\_died+=1  elif X\_train[i][2] == 1:    if(y\_train[i] == 1):  Female\_alive+=1  else:  Female\_died+=1  Accuracy\_Sex\_Male\_alive = Male\_alive/alive  Accuracy\_Sex\_Female\_alive = Female\_alive/alive  Accuracy\_Sex\_Male\_death = Male\_died/death  Accuracy\_Sex\_Female\_death = Female\_died/death  # tỉ lệ sống và chết của giới tính khách hàng  S = [[Accuracy\_Sex\_Male\_alive,Accuracy\_Sex\_Male\_death],  [Accuracy\_Sex\_Female\_alive,Accuracy\_Sex\_Female\_death]]    return H, S, A, R  pre = []  # hàm tính độ chính xác của tập test  def Predict(X\_test,y\_test,X\_train,y\_train):  global pre  H,S,A,R = Probability(X\_train,y\_train)  #hàm lấy số lượng y test  n = len(y\_test)  flag=0  for i in range(n):  temp\_no = [0,0,0]  temp\_yes = [0,0,0]  if(X\_test[i][0] == 0):  temp\_no[0] = H[0][1]  temp\_yes[0] = H[0][0]  elif(X\_test[i][0] == 1):  temp\_no[0] = H[1][1]  temp\_yes[0] = H[1][0]  elif(X\_test[i][0] == 2):  temp\_no[0] = H[2][1]  temp\_yes[0] = H[2][0]  elif(X\_test[i][0] == 3):  temp\_no[0] = H[3][1]  temp\_yes[0] = H[3][0]  if X\_test[i][1] == 0:  temp\_no[1] =A[0][1]  temp\_yes[1] = A[0][0]  elif X\_test[i][1] == 1:  temp\_no[1] =A[1][1]  temp\_yes[1] = A[1][0]  if X\_test[i][2] == 0:  temp\_no[2] = S[0][1]  temp\_yes[2] = S[0][0]  elif X\_test[i][2] == 1:  temp\_no[2] = S[1][1]  temp\_yes[2] = S[1][0]  ac = temp\_yes[0]\*temp\_yes[1]\*temp\_yes[2]\*R[0]  bc = temp\_no[0]\*temp\_no[1]\*temp\_no[2]\*R[1]  Px = ac + bc  Pyes = ac/Px  Pno = bc/Px  temp\_ytest = -1    if(Pyes > Pno):  temp\_ytest = 1  else:  temp\_ytest = 0  pre.append(temp\_ytest)  # #Ghi dữ liệu y predict ra file để so sánh  # result\_naivebayes\_np.write(str(temp\_ytest) + "\n")  if temp\_ytest == y\_test[i]:  flag += 1  return flag/n  # result\_naivebayes\_np.close()  # in ra giá trị accuracy  print("Accuracy resulted by Predict = ", Predict(X\_test,y\_test,X\_train,y\_train))  #Tính F1\_Score - Độ đo hiệu quả của mô hình  def F1\_Score(pre, y\_test):  precision = 0 # = TP / (TP + FP)  recall = 0 # = TP / P  TP = 0  FP = 0  P = 0  F1 = 0  for i in range (0, len(y\_test)):  if pre[i] == 1 and y\_test[i] == 1:  TP += 1  elif pre[i] == 1 and y\_test[i] == 0:  FP += 1  precision = TP / (TP + FP)  print("Precision = ", precision)  for i in range (0, len(y\_test)):  if y\_test[i] == 1:  P += 1  recall = TP / P  print("Recall = ", recall)  F1 = 2\*precision\*recall/(precision + recall)  return F1  print("F1\_Score = ", F1\_Score(pre, y\_test)) |

### Decision\_Tree\_sk.py: *(Decision Tree dùng thư viện Sklearn)*

|  |
| --- |
| from Pre\_Processing\_Data import \*  from sklearn import tree  from sklearn.metrics import accuracy\_score  import numpy as np  # Đọc dữ liệu  X\_train, y\_train, X\_test, y\_test = Load\_Data('F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/My\_Data.txt')  #Phân loại bằng Decision Tree  clf = tree.DecisionTreeClassifier()  # sử dụng X\_train làm dữ liệu huấn luyện  # y\_train làm giá trị đích (label)  clf = clf.fit(X\_train, y\_train)  # hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X\_test  pre = clf.predict(X\_test)  # hàm tính toán độ chính xác y\_test/pre  accuracy = accuracy\_score(y\_test, pre)  # # Ghi dữ liệu y predict ra file để so sánh  # result\_decisiontree\_sk = open("F:/HK4/ML/Case\_Study\_#1/ML\_titanic-master-final/result\_decisiontree\_sk.txt", "a+")  # for i in range (0, len(pre)):  # result\_decisiontree\_sk.write(str(pre[i]) + "\n")  # result\_decisiontree\_sk.close()  #Tính F1\_Score - Độ đo hiệu quả của mô hình  def F1\_Score(pre, y\_test):  precision = 0 # = TP / (TP + FP)  recall = 0 # = TP / P  TP = 0  FP = 0  P = 0  F1 = 0  for i in range (0, len(y\_test)):  if pre[i] == 1 and y\_test[i] == 1:  TP += 1  elif pre[i] == 1 and y\_test[i] == 0:  FP += 1  precision = TP / (TP + FP)  print("Precision = ", precision)  for i in range (0, len(y\_test)):  if y\_test[i] == 1:  P += 1  recall = TP / P  print("Recall = ", recall)  F1 = 2\*precision\*recall/(precision + recall)  return F1  print("F1\_Score = ", F1\_Score(pre, y\_test))  # in ra giá trị accuracy  print("Accuracy resulted by sklearn = ", accuracy) |

## Demo kết quả chụp màn hình:

Hình IV.6



* **KNN\_sk.py**

Precision = 0.425

Recall = 0.5042372881355932

F1\_Score = 0.46124031007751937

Accuracy resulted by sklearn = 0.6217687074829932

* **KNN\_np.py**

Precision = 0.5333333333333333

Recall = 0.4745762711864407

F1\_Score = 0.5022421524663677

Accuracy resulted by sklearn = 0.6979591836734694

* **Naive\_Bayes\_sk.py**

Precision = 0.6666666666666666

Recall = 0.4830508474576271

F1\_Score = 0.5601965601965602

Accuracy resulted by sklearn = 0.7564625850340136

* **Naive\_Bayes\_np.py**

Accuracy resulted by Predict = 0.7646258503401361

Precision = 0.6981132075471698

Recall = 0.4703389830508475

F1\_Score = 0.5620253164556962

* **Decision\_Tree\_sk.py**

Precision = 0.9195402298850575

Recall = 0.3389830508474576

F1\_Score = 0.49535603715170273

Accuracy resulted by sklearn = 0.7782312925170068

# References (Tài liệu tham khảo)

1. <https://machinelearningcoban.com/2017/08/31/evaluation/#-precision-va-recall>
2. <https://machinelearningcoban.com/2017/01/08/knn/>
3. <https://machinelearningcoban.com/2018/01/14/id3/>
4. <https://scikit-learn.org/stable/modules/naive_bayes.html>
5. <https://vimentor.com/vi/lesson/22-gioi-thieu-numpy>
6. <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html>
7. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
8. <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html>
9. <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>
10. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
11. <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html>

**--//--**

**CASE STUDY #1**

**CLASSIFICATION – TITANIC DATASET**

Giảng viên: Huỳnh Thị Thanh Thương

Sinh viên thực hiện: 17520324 – Nguyễn Thành Danh

17521244 – Hồ Sỹ Tuyến

17520828 – Phan Nguyên