**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

---------- ☙★❧ ----------

# TÌM HIỂU THƯ VIỆN SCIKIT-LEARN

***Môn học:*** Trí tuệ nhân tạo

***Lớp:*** CS106.K21

***Giảng viên:*** ThS. Nguyễn Đình Hiển

***Nhóm sinh viên thực hiện:***

1. Lê Cao Hưng 17520539
2. Lê Phước Đạt 18520017
3. Phan Thanh Hải 18520705

**TP. HỒ CHÍ MINH, 02/07/2020**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**

---------- ☙★❧ ----------

# TÌM HIỂU THƯ VIỆN SCIKIT-LEARN

***Môn học:*** Trí tuệ nhân tạo

***Lớp:*** CS106.K21

***Giảng viên:*** ThS. Nguyễn Đình Hiển

***Nhóm sinh viên thực hiện:***

1. Lê Cao Hưng 17520539
2. Lê Phước Đạt 18520017
3. Phan Thanh Hải 18520705

**TP. HỒ CHÍ MINH, 02/07/2020**

# MỤC LỤC

# Chương 1. TỔNG QUAN VỀ THƯ VIỆN SCIKIT-LEARN

## 1.1. Bối cảnh ra đời

Scikit-learn ban đầu được đề xuất bởi David Cournapeau trong một dự án mùa hè của Google vào năm 2007. Later Matthieu Brucher tham gia dự án trên và bắt đầu sử dụng nó làm một phần luận văn tiến sĩ của ông ấy. Vào năm 2010, INRIA bắt đầu tài trợ và phiên bản đầu tiên được xuất bản (v0.1 beta) vào cuối tháng 1 năm 2010.

Dự án vẫn đang được nghiên cứu bởi một đội ngũ hơn 30 nhà nghiên cứu đến từ các công ty lớn INRIA, Google, Tinyclues và Python Software Foundation.

Năm 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort và Vincent Michel, tất cả từ Viện Nghiên cứu Khoa học Máy tính và Tự động hóa Pháp tại Rocquencourt, Pháp, đã lãnh đạo dự án và phát hành công khai đầu tiên vào ngày 1 tháng 2 năm 2010.

## 1.2. Công dụng của Scikit-learn

Scikit-learn (Sklearn) là thư viện mạnh mẽ nhất dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Nó có các thuật toán classification, regression và clustering khác nhau bao gồm các SVM, Random Forests, Naive Bayes Classification, K-means và Linear Regression, và được thiết kế để tương tác với các thư viện khoa học và số của Python NumPy và SciPy.

Scikit-learn tích hợp tốt với nhiều thư viện Python khác, chẳng hạn như matplotlib và plotly cho plotting (phát họa sơ đồ), numpy cho xử lý mảng, pandas, scipy, và nhiều hơn nữa.

Scikit-learn phần lớn được viết bằng Python và sử dụng rộng rãi numpy cho các tính toán đại số tuyến tính và mảng với hiệu suất cao. Hơn nữa, một số thuật toán cốt lõi được viết bằng Cython để cải thiện hiệu suất. SVM được triển khai bởi trình bao bọc Cython xung quanh LIBSVM; hồi quy logistic và các máy vectơ hỗ trợ tuyến tính bằng một trình bao bọc tương tự xung quanh LIBLINEAR.

Thư viện được cấp phép bản quyền chuẩn FreeBSD và chạy được trên nhiều nền tảng Linux. Scikit-learn được sử dụng như một tài liệu để học tập.

# Chương 2. CÁCH CÀI ĐẶT THƯ VIỆN SCIKIT-LEARN

Để cài đặt scikit-learn trước tiên phải cài thư viện SciPy (Scientific Python). Những thành phần gồm:

- Numpy: Gói thư viện xử lý dãy số và ma trận nhiều chiều
- SciPy: Gói các hàm tính toán logic khoa học
- Matplotlib: Biểu diễn dữ liệu dưới dạng đồ thị 2 chiều, 3 chiều
- IPython: Notebook dùng để tương tác trực quan với Python
- SymPy: Gói thư viện các kí tự toán học
- Pandas: Xử lý, phân tích dữ liệu dưới dạng bảng

Những thư viện mở rộng của SciPy thường được đặt tên dạng SciKits. Như thư viện này là gói các lớp, hàm sử dụng trong thuật toán học máy thì được đặt tên là scikit-learn.

Scikit-learn hỗ trợ mạnh mẽ trong việc xây dựng các sản phẩm. Nghĩa là thư viện này tập trung sâu trong việc xây dựng các yếu tố: dễ sử dụng, dễ code, dễ tham khảo, dễ làm việc, hiệu quả cao.

Mặc dù được viết cho Python nhưng thực ra các thư viện nền tảng của scikit-learn lại được viết dưới các thư viện của C để tăng hiệu suất làm việc. Ví dụ như: Numpy(Tính toán ma trận), LAPACK, LibSVM và Cython.

# Chương 3. CÁC BỘ DATA PHỔ BIẾN VÀ CÓ SẴN TRONG SCIKIT-LEARN

## 3.1. Bộ dataset Iris flower

Đây là bộ data rất nổi tiếng bởi vì nó được sử dụng như bộ dữ liệu "hello world" trong machine learning.

Bộ dữ liệu gồm 150 quan sát về loài hoa, gồm 4 cột về kích thước về chiều dài, chiều rộng của cánh hoa và lá đài hoa.

Bộ dataset được sử dụng để làm bài tập machine learning cơ bản về phân loại hoa Iris.

## 3.2. Bộ dataset Digits

Đây là bộ data chứa các hình ảnh về chữ số viết tay gồm 10 lớp và mỗi lớp là 1 chữ số. Mỗi chữ số đều được số hóa dưới dạng ma trận 8x8.

Được sử dụng trong bài toán machine learning về phân loại chữ số viết tay.

## 3.3. Bộ dataset Boston house prices

Bộ data gồm 150 mẫu với 13 thuộc tính ảnh hưởng đến giá nhà ở thành phố Boston.

Thường được sử dụng trong bài toán regression về dự đoán giá nhà.

## 3.4. Bộ dataset Diabetes

Bộ data gồm 442 mẫu với 10 thuộc tính về các chỉ số của cơ thể.

Được sử dụng trong bài toán regression để ước tính mức độ thuyên giảm sau 1 năm điều trị.

## 3.5. Bộ dataset Wine recognition

Gồm 178 mẫu với 13 thuộc tính của rượu vang.

Được sử dụng trong bài toán classification để phân loại rượu vang.

### 3.6. Bộ dataset The 20newsgroups

Gồm 18000 bài đăng về 20 chủ đề tin tức. Module này có 2 bộ. Cái thứ nhất là *sklearn.datasets.fetch_20newsgroups* gồm dữ liệu thô chưa được feature engineering và cái thứ 2 là ***sklearn.datasets.fetch_20newsgroups_vectorize*** đã được feature engineering. Bộ dữ liệu này thường được sử dụng trong bài toán classification về phân loại chủ đề văn bản.

### 3.7. Bộ dataset California Housing

Bộ data gồm 20640 mẫu với 8 thuộc tính ảnh hưởng đến giá nhà ở bang California.

Thường được sử dụng trong bài toán regression về dự đoán giá nhà.

# Chương 4. XỬ LÝ DỮ LIỆU VÀ TRÍCH XUẤT ĐẶC TRƯNG VỚI SCIKIT-LEARN

## 4.1. Trích xuất đặc trưng từ file văn bản

### 4.1.1. Đặt vấn đề

Đối với các bài toán về xử lý văn bản thì các thuật toán machine learning không thể chạy trực tiếp trên văn bản được mà phải thực hiện ma trận hóa văn bản đó và sklearn có hỗ trợ các hàm xử lý văn bản cực kì hữu ích.

### 4.1.2. CountVectorizer()

#### 4.1.2.1. Chức năng

Hỗ trợ tiền xử lý dữ liệu văn bản, token hóa, lọc stop words:

#### 4.1.2.2. Cách thức thực hiện

Trước tiên tiến hành xây dựng mô hình bag of words (túi từ): xây dựng một tập từ điển các từ có trong văn bản sau đó với mỗi câu thứ *i*, đếm số lượng từ *w* và lưu trữ số lượng đếm được trong *X[i,j]* với *j* là số thứ tự của từ *w* ở trong từ điển. Từ đó ta được ma trận *X* là ma trận số hóa của văn bản.

Hàm **CountVectorizer()** có thể lựa chọn tham số *stop_words* để lọc ra các từ xuất hiện nhiều lần nhưng không có giá trị cao trong phân loại văn bản để tránh gây nhiễu cho model.

### 4.1.3. TfidfTransformer()

#### 4.1.3.1. Chức năng

Chuyển đổi từ ma trận kết quả của hàm **CountVectorizer()** (ma trận xây dựng bằng phương pháp đếm) sang ma trận về tần số xuất hiện. Bởi vì những tài liệu dài hơn thường có số lần đếm trung bình dài hơn so với tài liệu ngắn hơn tuy nhiên nó lại cùng một chủ đề nên phải thực hiện phương pháp này.

#### 4.1.3.2. Cách thức thực hiện

Công thức: *tfidf* = *tf* * *idf*, trong đó:

- *tf*: tần suất xuất hiện của một từ ở trong văn bản = số lần xuất hiện của từ đó / số lần xuất hiện của từ xuất hiện nhiều nhất trong văn bản.
- *idf*: nghịch đảo tần suất của văn bản = tổng số văn bản / số văn bản có chứa từ đang xét

#### 4.1.3.3. Ví dụ minh họa

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
sentence=['This is the first document.',
'This is the second second document.',
'And the third one.',
'Is this the first document?',]
vectorizer = CountVectorizer()
y = vectorizer.fit_transform(sentence)
print("Các từ trong đoạn văn:")
print(vectorizer.get_feature_names())
print("Ma trận hóa đoạn văn:")
print(y.toarray())
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(y).toarray()
print("tfidf:",tfidf)
```

Output:

```
Các từ trong đoạn văn:
['and', 'document', 'first', 'is', 'one', 'second', 'the',
'third', this']
```

```
Ma trận hóa đoạn văn:
[[0 1 1 1 0 0 1 0 1]
[0 1 0 1 0 2 1 0 1]
[1 0 0 0 1 0 1 1 0]
[0 1 1 1 0 0 1 0 1]]
Tfidf:
[[0. 0.43877674     0.54197657      0.43877674          0. 0.
0.35872874      0. 0.43877674]
[0. 0.27230147  0.       0.27230147 0. 0.85322574
0.22262429          0. 0.27230147]
[0.55280532     0. 0.              0. 0.55280532  0.
0.28847675          0.55280532 0.]
[0. 0.43877674  0.54197657         0.43877674          0. 0.
0.35872874          0. 0.43877674]]
```

### 4.2. Xử lý tính năng phân loại

#### 4.2.1. Đặt vấn đề

Giả sử chúng ta có một bộ dữ liệu về bài toán dự đoán giá nhà như trên:

```
data = [
{'price': 850000, 'rooms': 4, 'neighborhood': 'Queen Anne'},
{'price': 700000, 'rooms': 3, 'neighborhood': 'Fremont'},
{'price': 650000, 'rooms': 3, 'neighborhood': 'Wallingford'},
{'price': 600000, 'rooms': 2, 'neighborhood': 'Fremont'}
]
```

Việc lưu trữ dữ liệu như trên sẽ bị ngầm định rằng Queen Anne < Fremont < Wallingford < Fremont bởi vì cơ chế ánh xạ giá trị thông qua tính năng số học.

Để giải quyết vấn đề trên chúng ta cần phải thực hiện kĩ thuật one-hot encoding: bằng cách biến các giá trị cần chuyển đổi thành các vector nhị phân, bằng 1 nếu giá trị đó xuất hiện bằng 0 nếu không phải giá trị đó.

### 4.2.2. One-hot encoding

Kĩ thuật **one-hot encoding** đối với ví dụ ở trên:

- Đầu tiên ta sẽ xây dựng vector gồm các giá trị cần chuyển đổi:

```
vec=[‘Fremont’,’Queen Anne’,’Wallingford’]
```

- Mở rộng bộ dữ liệu thêm 3 cột tương ứng với số lượng các giá trị chuyển đổi. Duyệt từ trái sang phải của 3 cột vừa tạo ra, giá trị ở vị trí i sẽ bằng 1 nếu vec[i] bằng giá trị cần chuyển đổi của hàng đó.
- Kết quả:

```
array([[ 0, 1, 0, 850000, 4],
[ 1, 0, 0, 700000, 3],
[ 0, 0, 1, 650000, 3],
[ 1, 0, 0, 600000, 2]])
```

Kĩ thuật one-hot encoding có vẻ phức tạp tuy nhiên trong thư viện sklearn đã hỗ trợ hàm có sẵn để thực hiện kĩ thuật này: DictVectorizer()

### 4.2.3. Ví dụ minh họa

```
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False, dtype=int)
vec.fit_transform(data)
```

Kết quả:

```
array([[ 0, 1, 0, 850000, 4],
[ 1, 0, 0, 700000, 3],
```

```
[ 0, 0, 1, 650000, 3],
[ 1, 0, 0, 600000, 2]], dtype=int64)
```

Kĩ thuật one-hot encoding có một nhược điểm, nếu số lượng dữ liệu cần chuyển hóa lớn thì sẽ bùng nổ kích thước của bộ dữ liệu. Để giải quyết vấn đề này, người ta sẽ sử dụng kĩ thuật parse output.

Thư viện sklearn có hỗ trợ 2 công cụ **sklearn.preprocessing.OneHotEncoder** và **sklearn.feature_extraction.FeatureHasher** để hỗ trợ thực hiện kĩ thuật này.

### 4.3. Trích xuất đặc trưng hình ảnh

#### 4.3.1. Đặt vấn đề

Không giống như con người, máy tính chỉ hiểu được các hình ảnh dưới dạng các pixel, vì vậy trích xuất đặc trưng hình ảnh là điều không thể thiếu đối với các bài toán Machine Learning về xử lý ảnh.

#### 4.3.2. Histogram of Oriented Gradients (HOG)

Kĩ thuật HOG là kỹ thuật trích xuất đặc trưng ảnh cơ bản và được tích hợp sẵn trong thư viện sklearn (skimage).

#### 4.3.3. Các bước thực hiện

Kĩ thuật HOG thực hiện lần lượt các bước sau:

- Chuẩn hóa mức sáng và màu sắc của ảnh.
- Tính gradient của ảnh (đạo hàm ảnh).
- Chia ảnh thành các cell và xây dựng biểu đồ cường độ gradient của các pixel trong cell đó.
- Chuẩn hóa các block (các khối cell 2x2, 4x4,...).
- Trích vector đặc trưng của các block đã chuẩn hóa cho ảnh 64x128.

Những bước trên nghe có vẻ rất phức tạp tuy nhiên với sự hỗ trợ của **skimage** từ sklearn thì feature vector sẽ được trích xuất ra một cách dễ dàng.

### 4.4. Phát sinh tính năng

#### 4.4.1. Đặt vấn đề

Trong bài toán Linear Regression, có một vấn đề thường mắc phải là model tạo ra quá đơn giản so với bộ dữ liệu mục tiêu bởi vì sự thiếu hụt các tính năng. Để giải quyết điều đó, sklearn đã cung cấp hàm **PolynomialFeature()** để hỗ trợ trong việc thêm các tính năng từ tính năng gốc nhằm tăng độ phức tạp của model.

#### 4.4.2. Ví dụ minh họa

```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
x = np.array([1, 2, 3, 4, 5])
poly = PolynomialFeatures(degree=3, include_bias=False)
X2 = poly.fit_transform(x[:, np.newaxis])
print(X2)
```

Kết quả:

```
[[ 1. 1. 1.]
 [ 2. 4. 8.]
 [ 3. 9. 27.]
 [ 4. 16. 64.]
 [ 5. 25. 125.]]
```

Ở ví dụ trên với degree=3 nghĩa là chúng ta sẽ mở rộng ma trận đến bậc 3 và ta thấy được 2 cột đã thêm vào chính là bình phương giá trị cột đầu tiên và lập phương giá trị cột đầu tiên chỉ với 2 dòng lệnh.

### 4.5. Xử lí dữ liệu bị thiếu

#### 4.5.1. Đặt vấn đề

Ta thấy trong quá trình thu thập dữ liệu thì không phải lúc nào bộ dữ liệu chúng ta cũng rõ ràng, đồng nhất hay thậm chí còn thiếu hụt. Và để áp dụng được các thuật toán máy học thì việc xử lí dữ liệu thiếu hụt là điều kiện tiên quyết không thể thiếu. Để thuận tiện, sklearn đã cung cấp hàm **Imputer()** để giải quyết vấn đề này với nhiều chiến thuật giải quyết khác nhau.

### 4.5.2. Ví dụ minh họa

```
from numpy import np

from sklearn.preprocessing import Imputer

X = np.array([[ nan, 0, 3 ],  [ 3, 7, 9 ], [ 3, 5, 2 ], [ 4, nan, 6 ], [ 8, 8, 1 ]])

imp = Imputer(strategy='mean')

X2 = imp.fit_transform(X)
```

Kết quả:

```
array([[4.5, 0., 3.],

[ 3., 7., 9.],

[ 3., 5., 2.],

[ 4., 5., 6.],

[ 8., 8., 1.]])
```

Ở ví dụ trên ta thực hiện chiến lược điền vào chỗ trống bằng việc lấy trung bình cột có giá trị thiếu để điền vào. Ngoài ra còn một số chiến lược khác như: **median** (thay giá trị thiếu bằng giá trị ở giữa cột) và **most_frequent** (thay giá trị thiếu bằng giá trị xuất hiện nhiều nhất), chúng ta có thể thực hiện theo 2 chiến lược này bằng việc điều chỉnh tham số **strategy** ở hàm **Imputer()**.

## 4.6. Xây dựng đường ống tính năng

### 4.6.1 Đặt vấn đề

Với những hàm hỗ trợ của sklearn được nêu ở trên thì ta thấy mỗi công đoạn được giải quyết nhanh chóng tuy nhiên nếu thực hiện chuỗi các công đoạn thì còn khá là dài. Vì vậy sklearn đã hỗ trợ việc xây dựng một đường ống gồm chuỗi các công đoạn để máy tính tự xử lý, người lập trình chỉ việc xây dựng chuỗi các hàm theo đường ống đó.

### 4.6.2. Ví dụ minh họa

Giả sử chúng ta muốn xây dựng chuỗi công đoạn theo thứ tự:

- Điền giá trị thiếu.
- Thêm tính năng với cột bình phương.
- Huấn luyện model với thuật toán Linear Regression.

Ta chỉ cần thực hiện như sau:

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(Imputer(strategy='mean'),
                      PolynomialFeatures(degree=2),
                      LinearRegression()  )
```

Hoàn thành 3 bước chỉ với ba câu lệnh để thấy được sklearn mạnh mẽ đến mức nào.

# Chương 5. LỰA CHỌN MODEL VÀ HUẤN LUYỆN MODEL

## 5.1. Lựa chọn model

Có 2 loại kĩ thuật máy học phổ biến nhất hiện nay là **supervised learning** và **unsupervised learning**.

Trong **supervised learning** có 2 loại bài toán là **classification** và **regression**, trong **unsupervised learning** thường là bài toán về **clustering**.

Hiện nay có rất nhiều thuật toán về machine learning để giải quyết các bài toán này và sklearn cũng hỗ trợ hầu hết các thuật toán đó. Tuy nhiên nhóm chỉ báo cáo về 6 thuật toán cơ bản, thông dụng và phù hợp với cấp bậc sinh viên nhất, bao gồm:

- Multinomial Naive Bayes.
- Linear Regression.
- Support Vector Machines.
- Decision Trees.
- Random Forest.
- K-means Clustering.

### 5.1.1. Multinomial Naive Bayes

#### 5.1.1.1. Cơ sở lí thuyết

Thuật toán trên được xây dựng dựa trên lí thuyết Bayes, đó là một phương trình mô tả mối quan hệ về xác suất có điều kiện của các đối tượng thống kê. Thuật toán này thường được sử dụng trong bài toán về phân loại văn bản.

Định lí Bayes: $P(A|B) = \frac{P(B|A).P(A)}{P(B)}$

Model tính của Multinomial Naive Bayes khá phức tạp vì vậy để đơn giản hóa ta sẽ xem qua ví dụ về bài toán phân loại văn bản sau.

#### 5.1.1.2. Ví dụ minh họa

| Type | Doc | Word | Class |
|---|---|---|---|
| Training | 1 | Chinese Bejing Chinese | c |
| | 2 | Chinese Chinese Shangsai | c |
| | 3 | Chinese Macao | c |
| | 4 | Tokyo Japan Chinese | j |
| Test | 5 | Chinese Chinese Chinese Tokyo Japan | ? |

Xác suất văn bản là văn bản có class c: P(c) = $\frac{3}{4}$

Xác suất văn bản là văn bản có class j: P(j) = $\frac{1}{4}$

Thuật toán sẽ tiến hành tính toán xác suất xuất hiện mỗi từ trong bộ training với công thức sau: $P(word|class) = \frac{count(word,class)+1}{count(class)+|V|}$ với:

- *count(word, class)*: số lượng từ word của văn bản có trong nhãn class.
- *count(class)*: tổng số từ của văn bản có nhãn class.
- $|V|$: số từ khác nhau của toàn bộ văn bản.

$P(Chinese|c) = \frac{5+1}{8+6} = \frac{3}{7}$ ; $P(Tokyo|c) = \frac{1}{14}$ ; $P(Japan|c) = \frac{1}{14}$ ;

$P(Chinese|j) = \frac{2}{9}$ ; $P(Tokyo|j) = \frac{2}{9}$ ; $P(Japan|j) = \frac{2}{9}$ ;

Sau đó tính xác suất văn bản đối với từng loại nhãn và so sánh:

$P(c|d5) = \frac{3}{4} * \left(\frac{3}{7}\right)^3 * \frac{1}{14} * \frac{1}{14} = 0.0003$

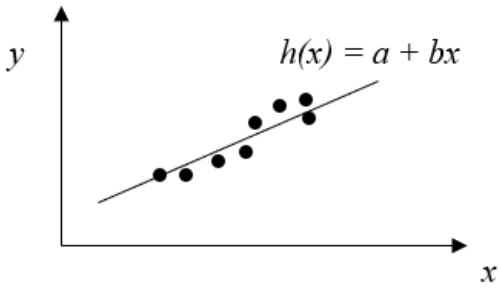
$P(j|d5) = \frac{1}{4} * \left(\frac{2}{9}\right)^3 * \frac{2}{9} * \frac{2}{9} = 0.0001$

Vì $P(c|d5) > P(j|d5)$ nên văn bản d5 thuộc lớp c.

### 5.1.2. Linear Regression

#### 5.1.2.1. Cơ sở lí thuyết

Đây là thuật toán phổ biến, đơn giản cho các bài toán về hồi quy về dự đoán giá nhà,giá chứng khoán. Ý tưởng chính của thuật toán chính tìm cách để tìm ra một đường đi qua các điểm dữ liệu có sẵn mà đạt được **cost function** là nhỏ nhất.




*Hình 5.1.*

#### 5.1.2.2. Cost function

Ta có thể hiểu **cost function** chính là trung bình bình phương khoảng biến thiên giữa tọa độ thực của một điểm so với tọa độ của điểm đó ở trên đường mà model huấn luyện.

Giả sử model chúng ta tìm được đường thẳng có phương trình như sau: $h(x) = a + bx$. Cost function sẽ được tính theo công thức:

$J(a,b) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$ , với:

- $m$: là số điểm dữ liệu .
- $h(x)$: tọa độ điểm i do đường thẳng biểu diễn.
- $y$: tọa độ thực của điểm i.

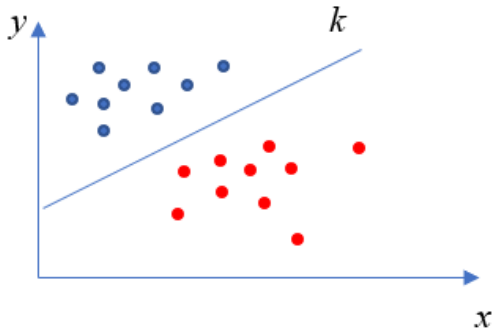
Để tìm ra các hệ số thích hợp $a$, $b$ trong phương trình đường $h(x)$ thì người ta sẽ sử dụng phương pháp **gradient descent**. Phương pháp này đơn giản là tận dụng sức mạnh tính toán của máy tính để dò tìm tham số cho hàm cost function cho tới khi hàm đạt được giá trị nhỏ nhất mà ta mong muốn.

### 5.1.3. Support Vector Machines (SVM)

#### 5.1.3.1. Đặt vấn đề

Là thuật toán linh hoạt, có thể sử dụng trong cả 2 loại bài toán classification và regression.

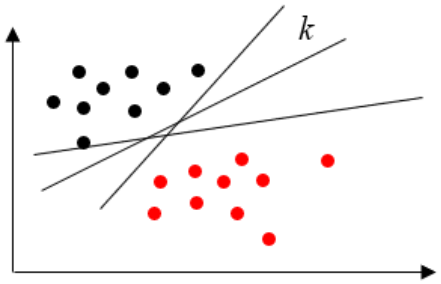
Ý tưởng chính của thuật toán này là tìm ra một đường thẳng hoặc một đường cong để có thể phân biệt ra các loại dữ liệu khác nhau.




*Hình 5.2.*

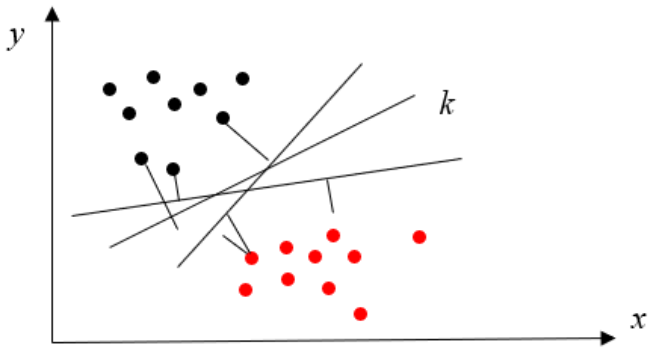
### 5.1.3.2. Các vấn đề cần giải quyết

Vấn đề đặt ra ở đây chúng ta có thể vẽ ra rất nhiều đường để phân chia 2 bộ dữ liệu như hình dưới đây. Vậy đường nào là đường phù hợp nhất chúng ta nên chọn ?



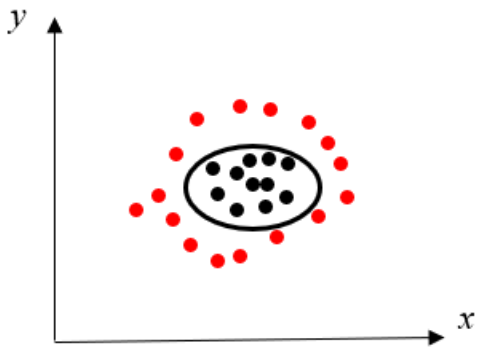

*Hình 5.3.*

SVM giải quyết việc này bằng cách vẽ thêm 2 đường lề từ các đường thẳng đến 2 điểm dữ liệu gần nhất của 2 lớp dữ liệu. Đường nào có khoảng lề lớn nhất chính là đường được chọn để phân chia bộ dữ liệu.




*Hình 5.4.*

Vậy đối với bộ dữ liệu không tuyến tính thì SVM sẽ thực hiện như thế nào? Đối với trường hợp này SVM sẽ sử dụng phép biến đổi kernel thành kernel RBF để xem xét các điểm dữ liệu dưới dạng không gian ba chiều và tìm ra quy luật tuyến tính của chúng.




*Hình 5.5.*

### 5.1.4. Decision Tree

#### 5.1.4.1. Cơ sở lí thuyết

Là một mô hình xây dựng theo cấu trúc cây trong đó các lá đại diện cho các phân loại còn cành đại diện cho các kết hợp của các thuộc tính dẫn tới phân loại đó.

Một cây quyết định có thể được học bằng cách chia tập hợp nguồn thành các tập con dựa theo một kiểm tra giá trị thuộc tính. Quá trình này được thực hiện đệ quy cho đến khi không còn phân tách được nữa

#### 5.1.4.2. Cách thức xây dựng

Thuật toán nổi tiếng để xây dựng cây quyết định là **Iterative Dichtomiser 3(ID3).** Các thuộc tính lựa chọn được đánh giá dựa trên **Hàm số Entropy.**

#### 5.1.4.3. Hàm số Entropy

Cho một phân phối xác suất của một biến rời rạc *x* có thể nhận *n* giá trị khác nhau x1,x2,…,xn. Giả sử xác suất để *x* nhận các giá trị này là $pi=p(x=x_i)$

Entropy của phân phối này là: $H(p) = -\sum_{i=1}^{n} pilog2(pi)$

Từ công thức trên ta thấy rằng hàm Entropy sẽ đạt giá trị nhỏ nhất nếu có một giá trị *pi*=1, đạt giá trị lớn nhất nếu tất cả các *pi* bằng nhau. Hàm Entropy càng lớn thì độ ngẫu nhiên của các biến rời rạc càng cao (càng không tinh khiết). Với cây quyết định, ta cần tạo cây để cho được lượng thông tin nhiều nhất hay Entropy cao nhất.

#### 5.1.4.4. Information Gain

Bài toán của ta trở thành, tại mỗi tầng của cây, cần chọn thuộc tính nào để độ giảm Entropy là thấp nhất.

Công thức tính: $Gain(S, f) = H(S) - H(f, S)$, với:

- H(S) là Entropy tổng của toàn bộ tập data set *S*.
- H(f, S) là Entropy được tính trên thuộc tính f.

Do H(S) là không đổi với mỗi tầng, ta chọn thuộc tính f có Entropy nhỏ nhất để thu được Gain(S, f) lớn nhất.

#### 5.1.4.5. Áp dụng lí thuyết vào ví dụ thực tế

Bài toán dự đoán nhu cầu mua ô tô. Ta có tập training data như bảng dưới:

| ID | Engine | Type | Color | 4WD | Want? |
|---|---|---|---|---|---|
| 1 | 2000cc | SUV | Silver | Yes | Yes |
| 2 | 1000cc | Sedan | Silver | Yes | Yes |
| 3 | 2000cc | Sport | Blue | No | No |
| 4 | 1000cc | SUV | Blue | No | Yes |
| 5 | 2000cc | Sedan | Silver | Yes | No |
| 6 | 2000cc | Sport | Blue | Yes | Yes |

| | | | | | |
|---|---|---|---|---|---|
| 7 | 1000cc | Sedan | Blue | No | Yes |
| 8 | 1000cc | SUV | Silver | No | Yes |

Xét thuộc tính Engine: thuộc tính này có 2 giá trị 1000cc và 2000cc. Gọi tập hợp điểm xếp lại theo thuộc tính Engine ta có 2 bảng nhỏ.

- Engine 1000cc (S1)

| ID | Engine | Type | Color | 4WD | Want? |
|---|---|---|---|---|---|
| 2 | 1000cc | Sedan | Silver | Yes | Yes |
| 4 | 1000cc | SUV | Blue | No | Yes |
| 7 | 1000cc | Sedan | Blue | No | Yes |
| 8 | 1000cc | SUV | Silver | No | Yes |

- Engine 2000cc (S2)

| ID | Engine | Type | Color | 4WD | Want? |
|---|---|---|---|---|---|
| 1 | 2000cc | SUV | Silver | Yes | Yes |
| 3 | 2000cc | Sport | Blue | No | No |
| 5 | 2000cc | Sedan | Silver | Yes | No |
| 6 | 2000cc | Sport | Blue | Yes | Yes |

- Child node ứng với Engine 1000cc sẽ có Entropy bằng 0 vì tất cả giá trị đều là Yes.

H(S1) = 0

$$H(S2) = \frac{-2}{4}\log(\frac{2}{4}) - \frac{-2}{4}\log(\frac{2}{4}) = 1$$

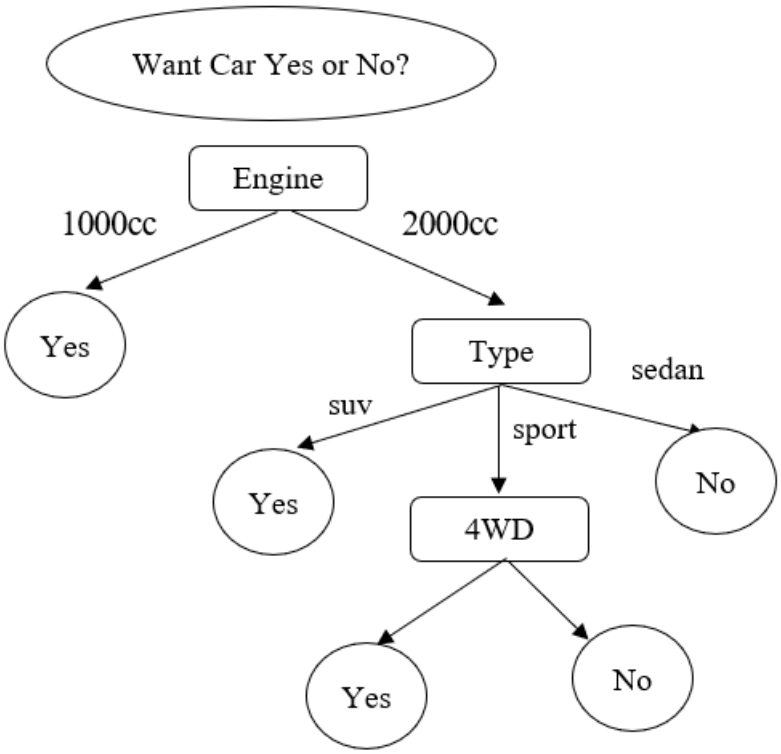
$$H(engine,S) = \frac{4}{8} H(S1) + \frac{4}{8} H(S2) = 0.5$$

Tính toán tương tự với thuộc tính Type, Color, 4WD ta được:

H(type, S) = 0.594 ; H(color, S) = 0.811; H(4wd, S) = 0.811

Vì thuộc tính Engine có Entropy nhỏ nhất nên ta chọn thuộc tính này là Node đánh giá đầu tiên. Với engine 1000cc, tất cả data đều có giá trị là Yes , vì vậy ta thu được node Yes ở nhánh 1000cc. Ở nhánh 2000cc ta tiếp tục xét với các thuộc tính còn lại với bộ data nhỏ hơn.

Kết quả:




*Hình 5.6.*

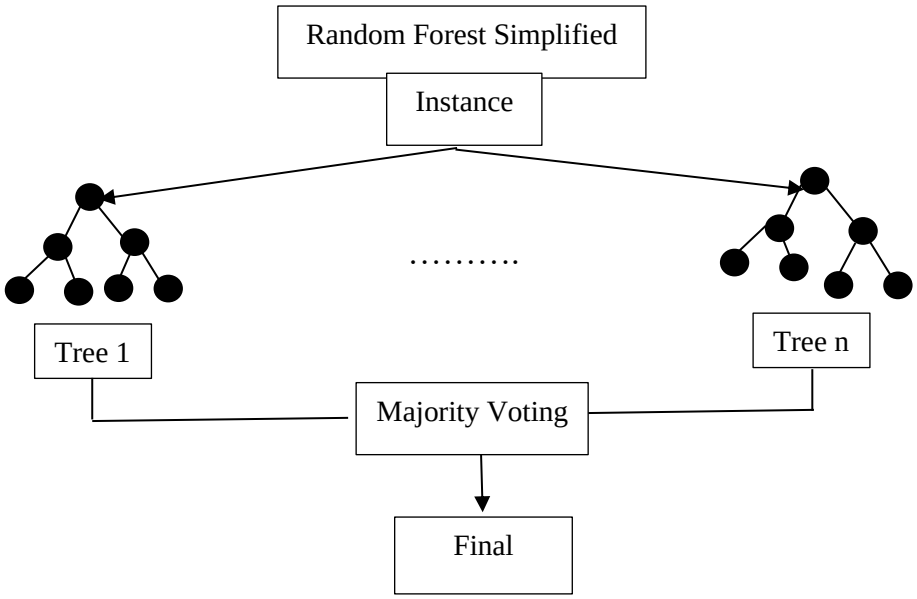
#### 5.1.4.6. Vấn đề overfitting

Đôi khi áp dụng thuật toán với bộ dữ liệu có nhiều dữ liệu nhiễu sẽ tạo thành cây quá sâu và mô hình tạo nên không thể rút ra được những đặc trưng tổng quát mà thay vào đó cố gắng phù hợp với từng chi tiết của bộ dữ liệu gây ra overfitting. Vì vậy ta cần hết sức chú ý vấn đề này.

### 5.1.5. Random forest

#### 5.1.5.1. Cơ sở lí thuyết

Đây là phương pháp xây dựng dựa trên một tập hợp Decision Tree và sử dụng phương pháp đánh giá để đưa ra quyết định về biến mục tiêu cần được dự báo. Thuật toán này có thể sử dụng cho cả 2 bài toán classification và regression




*Hình 5.7.*

### 5.1.5.2. Mã giả về cách thức xây dựng

Vì cấu trúc random forest quá phức tạp chúng ta không thể hiểu rõ được nên ta chỉ xét trên mã giả để hiểu rõ đôi chút về cách thực hoạt động của nó:

- Chọn ngẫu nhiên k features từ tập m features.
- Từ tập k features, tính toán ra node d là tốt nhất cho Node phân loại.
- Chia các node con theo node tốt nhất vừa tìm được.
- Lặp lại bước 1 cho đến khi đạt đến k node.
- Lại lại bước 1-4 để tạo ra n cây.

### 5.1.5.3. Cách thức hoạt động

Để biểu diễn dự đoán sử dụng Random Forest để huấn luyện, ta sử dụng các bước bên dưới:

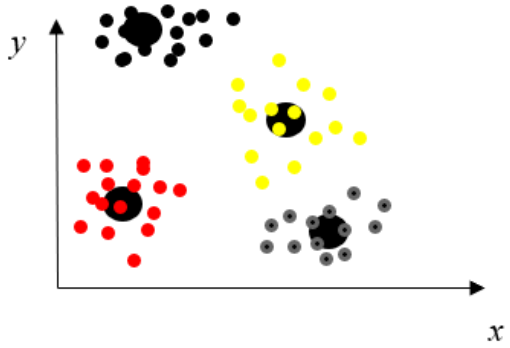
- Lấy các test features và sử dụng cây quyết định để tạo ra dự đoán kết quả, lưu nó vào một danh sách.
- Tính toán số lượng vote trên toàn bộ Forest cho từng kết quả.
- Lấy kết quả số lượng vote lớn nhất làm kết quả cuối cho mô hình.

### 5.1.6. K-means clustering

### 5.1.6.1. Cơ sở lý thuyết

Thuật toán k-means là thuật toán cơ bản dùng cho bài toán thuộc unsupervised learning. Thuật toán này tìm kiếm một số cụm dữ liệu được xác định trước trong bộ dữ liệu không được gán nhãn theo cơ chế đơn giản sau:

- Trung tâm của mỗi cụm bằng trung bình số học của tất cả nhưng điểm thuộc cụm.
- Mỗi điểm sẽ gần trung tâm cụm của nó hơn so với trung tâm cụm khác.




*Hình 5.8.*

Để phân ra các cụm trên bộ dữ liệu đã cho thì k-means đã sử dụng một cách tiếp cận gọi là **Expectation – Maximization**.

#### 5.1.6.2. Expectation – Maximization (tối đa hóa kì vọng)

Đoán một số cụm trung tâm.

Lặp lại cho đến khi hội tụ.

- Expection-step: Gán các điểm cho trung tâm cụm gần nó nhất.
- Maximization-step: Thiết lập giá trị cụm trung tâm lại bằng trung bình các điểm.

#### 5.1.6.3. Hạn chế của thuật toán K-means

Thuật toán không thể tự tính toán được bao nhiêu cụm cần phân tách mà chúng ta phải khai bao tham số **n_cluster** cho nó và thuật toán sẽ tìm ra cách phân chia tối ưu nhất.

Trong dữ liệu thực tế thì khoảng giữa các cụm giữ liệu thường là tuyến tính cho nên thuật toán sẽ có thể sai trong vấn đề phân cụm. Để giải quyết trường hợp người ta sử dụng phương pháp chuyển đổi kernel (đề cập ở Support Vector Machine) với hàm SpectralClustering() được import từ module sklearn.cluster

Bởi vì cơ chế truy cập mọi điểm dữ liệu trên mỗi vòng lặp cho nên nếu bộ dữ liệu quá lớn thì thời gian thực thi sẽ rất chậm.

## 5.2. Huấn luyện model

### 5.2.1. Đặt vấn đề

Ở phần trên chúng ta đã đề cập về cơ sở lý thuyết, cơ chế hoạt động và một số hạn chế của các thuật toán được tiến hành nghiên cứu. Ý tưởng không quá phức tạp tuy nhiên việc viết chương trình thì không bao giờ là dễ dàng. Để giải quyết vấn đề đó, chúng ta đã có sự hỗ trợ cực kì mạnh mẽ từ sklearn với việc chỉ cần import model và hàm fit() với bộ dữ liệu đã cho.

### 5.2.2. Ví dụ minh họa

```
from sklearn.svm import SVC

from sklearn.naive_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make_classification


X, y = make_classification (n_samples = 1000, n_features = 4,
n_informative = 2, n_redundant = 0)

SVC_model = SVC()

MNB_model = MultinomialNB()

Dtree_model = DecisionTreeClassifier(max_depth=2)

RF_model  =  RandomForestClassifier(max_depth=2,random_state=0)


SVC_model.fit(X,y)

MNB_model.fit(X,y)
```

```
Dtree_model.fit(X,y)
```

```
RF_model.fit(X,y)
```

Chỉ cần import và hàm **fit()** sẽ giúp ta thực hiện huấn luyện model một cách đơn giản mà không cần làm bất cứ điều gì khác.

# Chương 6. ĐÁNH GIÁ MODEL VÀ ĐIỀU CHỈNH SIÊU THAM SỐ

## 6.1. Đánh giá model là gì?

Đánh giá model đơn giản là tiến hành dự đoán bằng model đã được train rồi so sánh kết quả giữa kết quả model dự đoán và kết quả thực tế. Tuy nhiên nếu chúng ta sử dụng lại bộ dữ liệu đã train để đánh giá hiệu suất model thì là một phương pháp hoàn toàn sai vì trong quá trình học tập model đã tiến hành ghi nhớ đặc điểm của bộ dữ liệu đó.

## 6.2. Một số cách đánh giá model phổ biến

### 6.2.1. Giữ lại tập con của bộ dữ liệu

Sử dụng hàm **train_test_split()** để phần chia bộ dữ liệu thành 2 tập với tỉ lệ tùy chọn.

Ví dụ: Chia bộ dữ liệu theo tỉ lệ 50-50

```
from sklearn.cross_validation import train_test_split

x1,x2,y1,y2 = train_test_split(x,y,random_state = 1, train_size = 0.5)

model.fit(x1,y1)

y2_model = model.predict(x2)
```

Một bất lợi của phương pháp giữ lại tập con chính là mất đi một phần dữ liệu để huấn luyện. Nếu bộ dữ liệu ít thì điều này cực kì bất lợi. Vì vậy người ta nghĩ ra một phương gọi là đánh giá chéo.

### 6.2.2. Đánh giá chéo

Ý tưởng chính của phương pháp này là chia mẫu dữ liệu ban đầu thành *n* mẫu con để cho việc training chỉ thực hiện trên mẫu con đơn, các mẫu con còn lại dùng cho việc xác nhận và kiểm chứng lại phân tích đầu tiên đó.

Sử dụng hàm **cross_val_score()** để chia bộ dữ liệu và đánh giá chéo

Ví dụ:

```
from sklearn.cross_validation import cross_val_score

cross_val_score(model,x,y,cv=5)
```

### 6.2.3. Điều chỉnh siêu tham số

Đây là bước khá quan trọng nếu muốn model đạt hiệu suất cao hơn vì việc điều chỉnh một vài tham số ở một số bước sẽ giúp model phù hợp hơn với bộ dữ liệu.

Chúng ta chỉ cần lựa chọn tham số điều chỉnh còn việc chọn lựa tham số để đạt hiệu suất cao nhất thì đã có sklearn hỗ trợ với phương pháp **Grid Search**.

Ví dụ: Về tinh chỉnh tham số trên model có pipeline như trên:

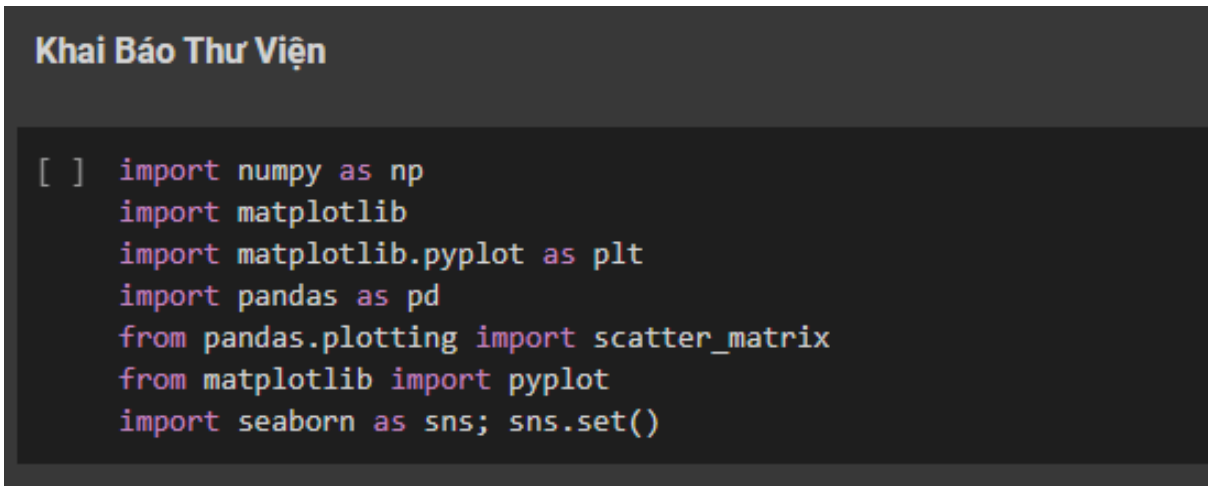
```
from sklearn.grid_search import GridSearchCV

param_grid = { ‘polynomialfeatures_degree’:np.arrange(21),

 ‘linearregression_fit_intercept’:[True,False],

 ‘linearregression_normalize’:[True,False],

}

Grid = GridSearch(PolynomialRegression(),param_grid,cv=7)

Grid.fit(X,y)

Grid.best_params_ #in ra danh sách tham số được chọn.
```

# Chương 7. ỨNG DỤNG VÀ KẾT QUẢ

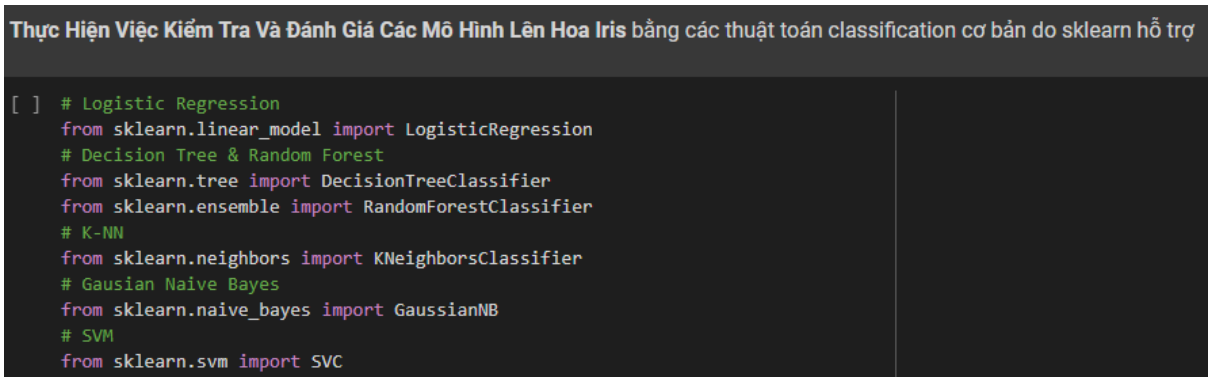
Đề tài thực hiện mà nhóm chúng em hướng tới là xây dựng 1 demo để thể hiện khả năng hỗ trợ của sklearn với các bài toán về Machine Learning.

Trong demo, chúng em sử dụng 6 thuật toán thường dùng để giải quyết về bài toán phân loại hoa Iris trong Machine Learning, là một tập dataset kinh điển của Sklearn.

Bài toán hoa Iris được phân loại dựa vào các thông số về độ dài của các phần của hoa Iris mà đưa ra kết quả




*Hình 7.1. Khai báo thư viện*




*Hình 7.2. Khai báo mô hình*

```
# Ma Trận Dữ Liệu và 5 Phần Tử Đầu
print(dataset.shape)
print(dataset)
# Mô Tả Dữ Liệu
```

```
(150, 5)
     sepal-length  sepal-width  petal-length  petal-width          class
0             5.1          3.5           1.4          0.2    Iris-setosa
1             4.9          3.0           1.4          0.2    Iris-setosa
2             4.7          3.2           1.3          0.2    Iris-setosa
3             4.6          3.1           1.5          0.2    Iris-setosa
4             5.0          3.6           1.4          0.2    Iris-setosa
..            ...          ...           ...          ...            ...
145           6.7          3.0           5.2          2.3  Iris-virginica
146           6.3          2.5           5.0          1.9  Iris-virginica
147           6.5          3.0           5.2          2.0  Iris-virginica
148           6.2          3.4           5.4          2.3  Iris-virginica
149           5.9          3.0           5.1          1.8  Iris-virginica

[150 rows x 5 columns]
```

*Hình 7.3. Mô tả cấu trúc của dữ liệu*

```
def LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scalin
g=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)
```

View source

Logistic Regression (aka logit, MaxEnt) classifier.
In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)
This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers.

*Hình 7.4. Logistic Regression*

```
def KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric
='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

View source

Classifier implementing the k-nearest neighbors vote.
Read more in the :ref: User Guide <classification> .

*Hình 7.5. K-NN*

```
def RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=2, min_
samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_dec
rease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, ver
bose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

View source

A random forest classifier.
A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).
Read more in the :ref: User Guide <forest> .

*Hình 7.6. Random Forest*

```
def DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0)
```

View source

A decision tree classifier.
Read more in the :ref: User Guide <tree> .

*Hình 7.7. Decision Tree*

```
def GaussianNB(priors=None, var_smoothing=1e-09)
```

View source

Gaussian Naive Bayes (GaussianNB)
Can perform online updates to model parameters via :meth: partial_fit . For details on algorithm used to update feature means and variance online, see Stanford CS tech report STAN-CS-79-773 by Chan, Golub, and LeVeque:
http://i.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf
Read more in the :ref: User Guide <gaussian_naive_bayes> .

*Hình 7.8. Naïve Bayes*

```
def SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)
```

View source

C-Support Vector Classification.
The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using :class: sklearn.svm.LinearSVC or :class: sklearn.linear_model.SGDClassifier instead, possibly after a :class: sklearn.kernel_approximation.Nystroem transformer.
The multiclass support is handled according to a one-vs-one scheme.
For details on the precise mathematical formulation of the provided kernel functions and how gamma , coef0 and degree affect each other, see the corresponding section in the narrative documentation: :ref: svm_kernels .
Read more in the :ref: User Guide <svm_classification> .

*Hình 7.9. SVM*

Đánh giá f1-score

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(Y_validation, predictions))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Iris-setosa | 1.00 | 1.00 | 1.00 | 19 |
| Iris-versicolor | 0.93 | 1.00 | 0.97 | 14 |
| Iris-virginica | 1.00 | 0.91 | 0.95 | 11 |
| accuracy | | | 0.98 | 44 |
| macro avg | 0.98 | 0.97 | 0.97 | 44 |
| weighted avg | 0.98 | 0.98 | 0.98 | 44 |

*Hình 7.10. Tỷ lệ huấn luyện model*

```
def Predict_Func(x,y,z,t,model=tuning_model,train=X_train):
    arr=[[x,y,z,t]]
    pred=model.predict(arr)
    print("Loai hoa duoc model du doan la: ",pred[0])
    print("Thank you")
```

```
hide_data
```

| | sepal-length | sepal-width | petal-length | petal-width | class |
|---|---|---|---|---|---|
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

```
Predict_Func(6.5,3.0,5.2,2.0)
```

```
Loai hoa duoc model du doan la:  Iris-virginica
Thank you
```

*Hình 7.11. Chạy thử model (sử dụng SVM)*

Đánh giá f1-score

```
[ ]  from sklearn.metrics import classification_report
     print(classification_report(Y_validation, predictions))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Iris-setosa | 1.00 | 1.00 | 1.00 | 19 |
| Iris-versicolor | 0.93 | 1.00 | 0.97 | 14 |
| Iris-virginica | 1.00 | 0.91 | 0.95 | 11 |
| accuracy | | | 0.98 | 44 |
| macro avg | 0.98 | 0.97 | 0.97 | 44 |
| weighted avg | 0.98 | 0.98 | 0.98 | 44 |

*Hình 7.12. Kết quả*

Bộ công cụ đã dùng cho thấy tỷ lệ dự đoán rất tốt đạt được 96,6% là độ chính xác rất cao khi sử dụng thuật toán SVM để huấn luyện model.

# TÀI LIỆU THAM KHẢO


**Tiếng Việt**

[1]. Thư Viện Scikit-learn Trong Python Là Gì?, https://codelearn.io/sharing/scikit-learn-trong-python-la-gi, ngày 02/07/2020

**Tiếng Anh**

[2]. Jeff Delaney (2016), 10 Classifier Showdown in Scikit-Learn, https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn, ngày 02/07/ 2020

[3]. Henry Badgery (2020), Project of the Month: scikit-learn, https://medium.com/openteams/project-of-the-month-scikit-learn-707138ec5f59, ngày 02/07/2020

[4]. An introduction to machine learning with scikit-learn, https://scikit-learn.org/stable/tutorial/basic/tutorial.html, ngày 02/07/2020

[5]. Installing scikit-learn, https://scikit-learn.org/stable/install.html, ngày 02/07/2020

[6]. Related Projects, https://scikit-learn.org/stable/related_projects.html, ngày 02/07/2020