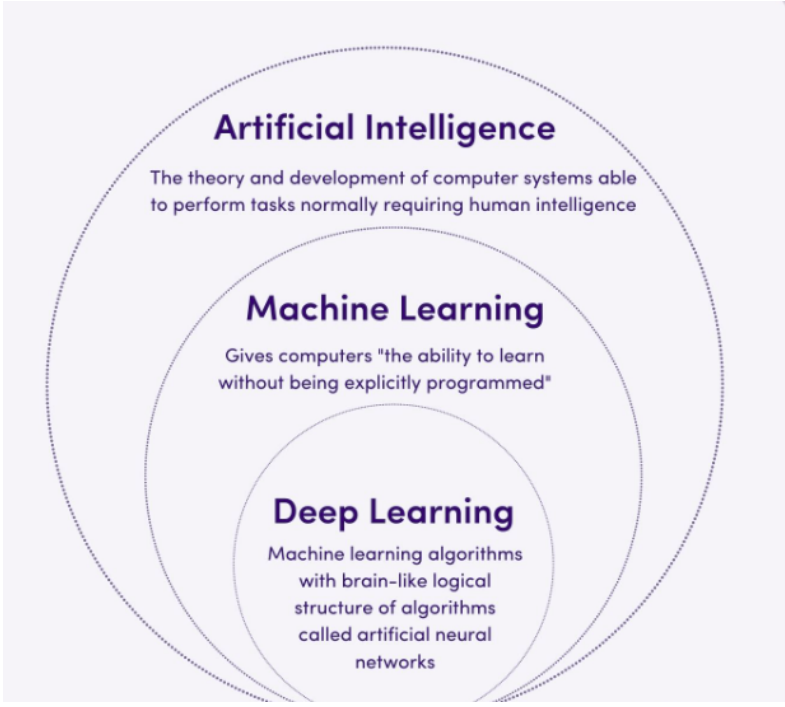
CHƯƠNG 4: HỌC MÁY

Môn học: Nhập môn Khoa học Dữ liệu

Giảng viên: Nguyễn Kiều Linh
Email: linhnk@ptit.edu.vn
Học viện Công nghệ Bưu chính Viễn thông

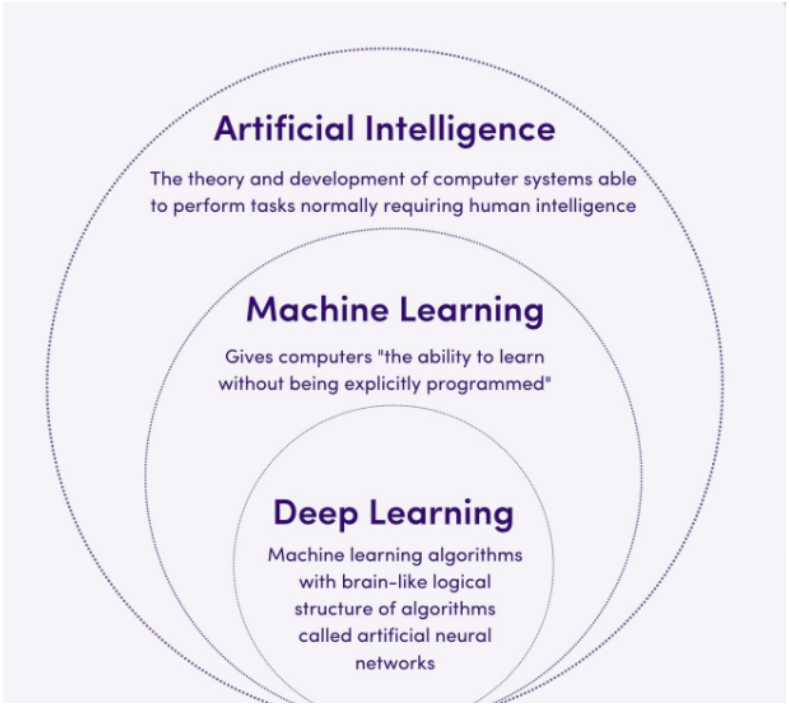
Hà Nội, năm 2023
http://www.ptit.edu.vn

Học máy là gì?



Học máy hay máy học (Machine learning) là một nhánh của trí tuệ nhân tạo (AI), nó là một lĩnh vực nghiên cứu cho phép máy tính có khả năng cải thiện chính bản thân chúng dựa trên dữ liệu mẫu (training data) hoặc dựa vào kinh nghiệm (những gì đã được học)

Học máy là gì?



Machine learning có thể tự dự đoán hoặc đưa ra quyết định mà không cần được lập trình cụ thể.

Phân loại Machine learning

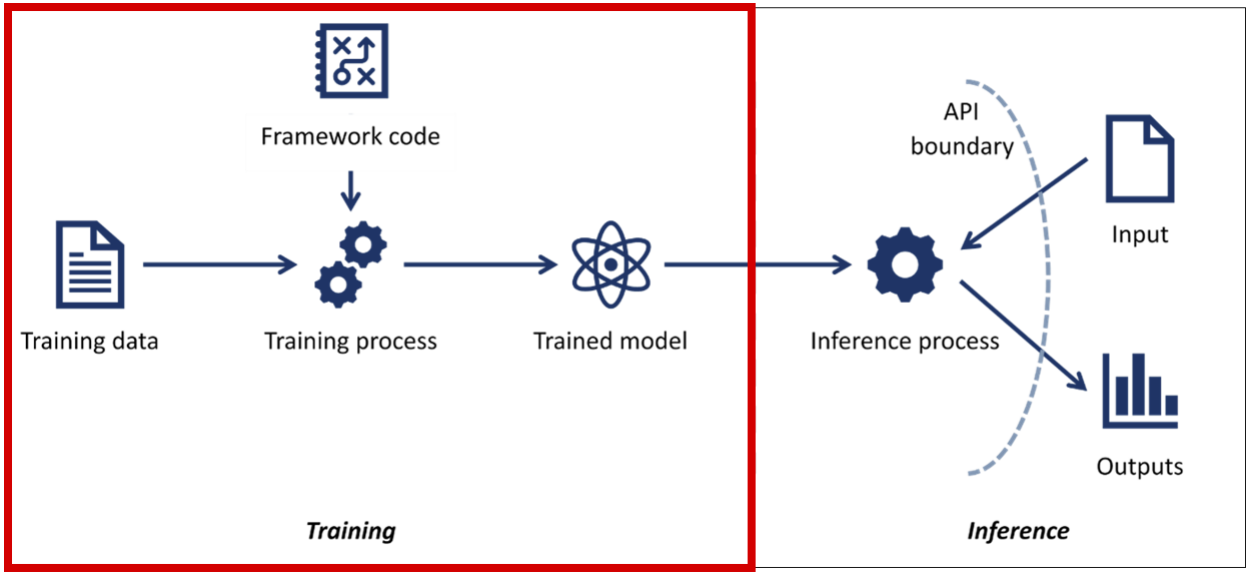
Có rất nhiều cách phân loại machine learning, phân loại theo phương thức học machine learning sẽ được phân thành các loại chính sau:

- Supervised learning: học có giám sát
- Unsupervised learning: học không giám sát
- Semi-Supervised Learning: học bán giám sát
- Reinforcement Learning: Học Củng Cố

Nội dung chính

Nội dung chính

Học và suy diễn



Quá trình dạy một thuật toán để tạo dự đoán hoặc thực hiện hành động dựa trên dữ liệu đầu vào được gọi là **đào tạo - training**. Khi đó các hệ thống máy tính "học"tự động từ dữ liệu để giải quyết những vấn đề cụ thể. Ví dụ như các máy có thể "học"cách phân loại thư điện tử xem có phải thư rác (spam) hay không và tự động xếp thư vào thư mục tương ứng.

Nội dung chính

Đánh giá mô hình

Sau khi đào tạo mô hình, bước tiếp theo trong Machine Learning là **đánh giá mô hình** mới tạo. Tùy thuộc vào các loại phép đo khác nhau mà mô hình huấn luyện được đánh giá là tốt hay không tốt. Về cơ bản, độ chính xác của mô hình huấn luyện đạt trên 80% được coi là đảm bảo tính hiệu quả.

Tại sao cần đánh giá mô hình?

Việc đánh giá mô hình giúp chúng ta giải quyết những vấn đề sau:

- Mô hình đã được huấn luyện thành công hay chưa?
- Mức độ thành công của mô hình tốt đến đâu?
- Khi nào nên dừng quá trình huấn luyện?
- Khi nào nên cập nhật mô hình?

Trả lời được 4 câu hỏi trên, chúng ta có thể quyết định mô hình này có thực sự phù hợp cho bài toán hay không.

Độ đo khi đánh giá mô hình

Để có thể áp dụng đúng thước đo đánh giá mô hình phù hợp, chúng ta cần hiểu bản chất, ý nghĩa cũng như các trường hợp sử dụng nó. Bài giảng giới thiệu một số độ đo đánh giá đối với mô hình phân loại và mô hình hồi quy.

Độ đo cho mô hình phân loại

Khi thực hiện bài toán phân loại, có 4 trường hợp của dự đoán có thể xảy ra:

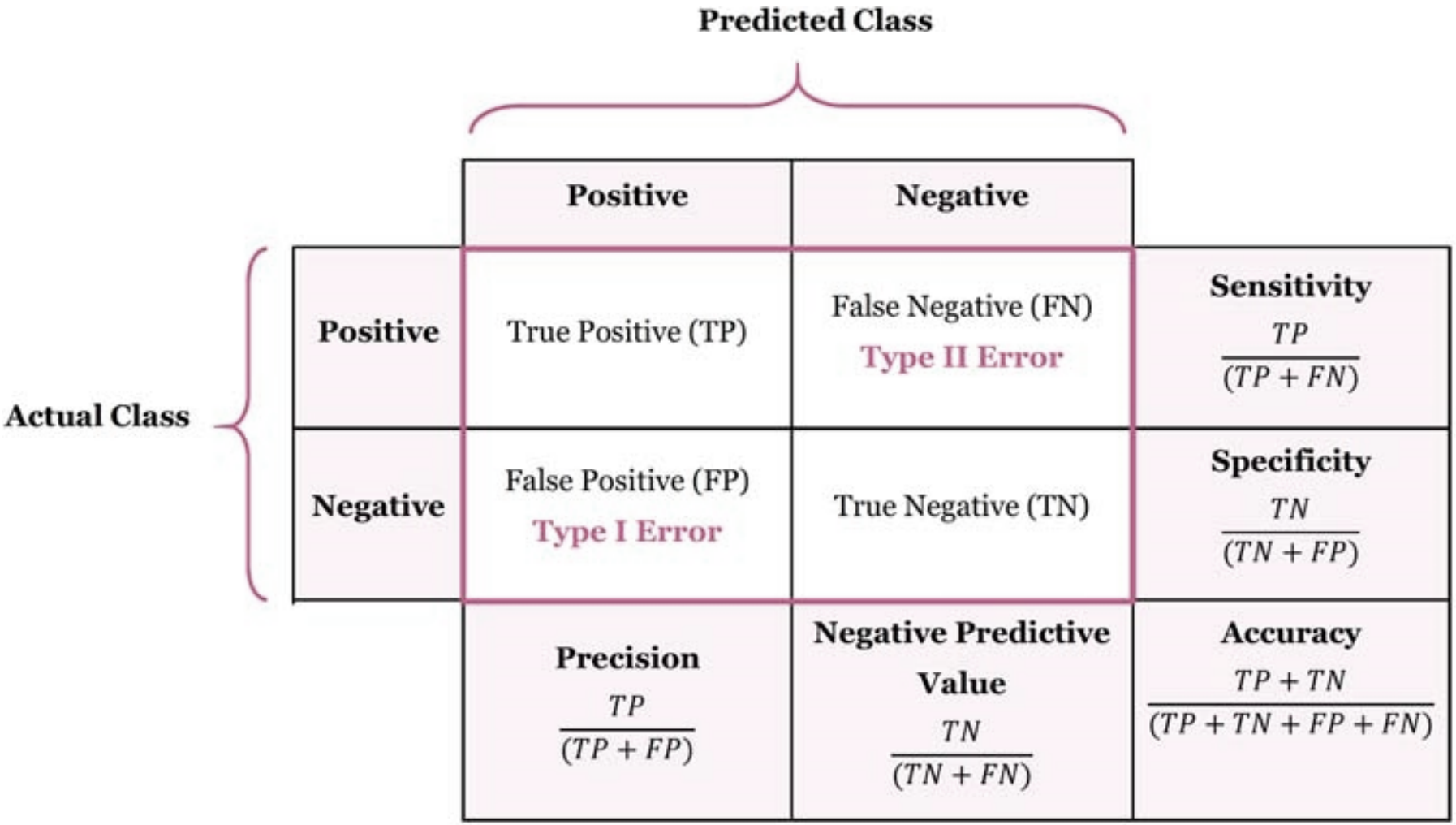
- True Positive (TP): đối tượng ở lớp Positive, mô hình phân đối tượng vào lớp Positive (dự đoán đúng)
- True Negative (TN): đối tượng ở lớp Negative, mô hình phân đối tượng vào lớp Negative (dự đoán đúng)
- False Positive (FP): đối tượng ở lớp Negative, mô hình phân đối tượng vào lớp Positive (dự đoán sai) – Type I Error
- False Negative (FN): đối tượng ở lớp Positive, mô hình phân đối tượng vào lớp Negative (dự đoán sai) – Type II Error

Độ đo cho mô hình phân loại

Ví dụ: Phân loại 1100 ảnh có phải mèo hay không, trong dữ liệu dự đoán có 100 ảnh là mèo (cat), 1000 ảnh không phải là mèo (non-cat). Ở đây, kết quả dự đoán là như sau

- Trong 100 ảnh mèo dự đoán đúng 90 ảnh, còn 10 ảnh được dự đoán là không phải. Cat là "positive" và non-cat là "negative",
  - 90 ảnh được dự đoán là cat, được gọi là True Positive,
  - còn 10 ảnh được dự đoán non-cat kia được gọi là False Negative
- Trong 1000 ảnh non-cat,
  - dự đoán đúng được 940 ảnh là non-cat, được gọi là True Negative,
  - còn 60 ảnh bị dự đoán nhầm sang cat được gọi là False Positive

Confusion matrix



Độ đo Accuracy

Accuracy được định nghĩa là tỷ lệ phần trăm dự đoán đúng cho dữ liệu thử nghiệm. Nó có thể được tính toán dễ dàng bằng cách chia số lần dự đoán đúng cho tổng số lần dự đoán.

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

Tức là

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

Độ đo Accuracy

Áp dụng vào bài toán Cat/Non-cat, tính Accuracy?

Độ đo Accuracy

Áp dụng vào bài toán Cat/Non-cat, tính Accuracy?

$$\text{Accuracy} = \frac{90 + 940}{1000 + 100} = 93.6\%$$

Độ đo Accuracy

Áp dụng vào bài toán Cat/Non-cat, tính Accuracy?

$$\text{Accuracy} = \frac{90 + 940}{1000 + 100} = 93.6\%$$

Nhược điểm của cách đánh giá này là chỉ cho ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào. Sẽ có rất nhiều trường hợp thước đo Accuracy không phản ánh đúng hiệu quả của mô hình.
Giả sử mô hình dự đoán tất cả 1100 ảnh là Non-cat, thì Accuracy vẫn đạt tới $1000/1100 = 90.9\%$.

Độ đo Precision

Precision sẽ cho chúng ta biết thực sự có bao nhiêu dự đoán Positive là thật sự True. Tức là

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Áp dụng vào bài toán Cat/Non-cat, tính Precision(cat) và Precision(non-cat)?

Độ đo Precision

Precision sẽ cho chúng ta biết thực sự có bao nhiêu dự đoán Positive là thật sự True. Tức là

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Áp dụng vào bài toán Cat/Non-cat, tính Precision(cat) và Precision(non-cat)?

$$\text{Precision(cat)} = \frac{90}{90 + 60} = 60\%,$$

$$\text{Precision(non-cat)} = \frac{940}{940 + 10} = 98.9\%.$$

Độ đo Recall

Recall cũng là một độ đo quan trọng, nó đo lường tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive. Công thức của Recall như sau:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Áp dụng vào bài toán Cat/Non-cat, tính Recall(cat) và Recall(non-cat)?

Độ đo Recall

Recall cũng là một độ đo quan trọng, nó đo lường tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive. Công thức của Recall như sau:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

Áp dụng vào bài toán Cat/Non-cat, tính Recall(cat) và Recall(non-cat)?

$$\text{Recall(cat)} = \frac{90}{90 + 10} = 90\%,$$

$$\text{Recall(non-cat)} = \frac{940}{940 + 60} = 94\%.$$

Recall cao đồng nghĩa với việc True Positive Rate cao, tức là tỷ lệ bỏ sót các điểm thực sự là positive là thấp

Điểm F-score

Precision và Recall rất hữu ích trong trường hợp các lớp không được phân bổ đồng đều. Vì vậy, phải đánh giá cả Precision và Recall của một mô hình đưa ra kết luận chính xác. Để có thể kết hợp giữa Precision và Recall, chúng ta có thể tính điểm F-score.

$$F_\beta = (1 + \beta^2) \frac{\text{Precision} * \text{Recall}}{(\beta^2 * \text{Precision}) + \text{Recall}}$$

Tham số β cho phép chúng ta kiểm soát sự cân bằng giữa Precision và Recall.

- $\beta < 1$ tập trung nhiều hơn vào Precision.
- $\beta > 1$ tập trung nhiều hơn vào Recall.
- $\beta = 1$ tập trung vào cả Precision và Recall.

Điểm F1-score

Khi $\beta = 1$, ta sử dụng F1-score, là kỳ vọng harmonic (harmonic mean) của Precision và Recall. F1-score lớn khi cả 2 giá trị Precision và Recall đều lớn. Ngược lại, chỉ cần 1 giá trị nhỏ sẽ làm cho F1-Score nhỏ.

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-Score càng lớn càng tốt. Khi lý tưởng nhất thì F1-score = 1 (khi Recall = Precision=1).

★ Áp dụng vào bài toán Cat/Non-cat, tính F1-Score và nhận xét?

Đánh giá Mô hình Hồi quy

- Các độ đo cho mô hình hồi quy khá khác so với các độ đo của mô hình phân loại vì phải dự đoán trong một khoảng liên tục thay vì một số các lớp rời rạc.
- Ví dụ xây dựng một mô hình dự đoán giá của một ngôi nhà là 2 tỷ đồng nhưng nó bán được 2,1 tỷ đồng thì đó được coi là mô hình tốt, trong khi bài toán phân loại chỉ quan tâm xem ngôi nhà đấy có được bán với giá 2 tỷ hay không.

Mean squared error (MSE)

Mean squared error được định nghĩa MSE được định nghĩa là trung bình tổng bình phương sai số giữa đầu ra dự đoán và kết quả thực. Mean squared error thường được sử dụng vì nó không thể xác định được liệu dự đoán quá cao hay quá thấp, nó chỉ báo cáo rằng dự đoán không chính xác.

$$MSE = \frac{1}{n}\sum_{1}^{n}(y_i - \hat{y}_i)^2.$$

MSE có miền giá trị từ $[0, +\infty]$. Trên cùng tập dữ liệu, MSE càng nhỏ thì có độ chính xác càng cao. Tuy nhiên, vì lấy bình phương sai số nên đơn vị của MSE khác với đơn vị của kết quả dự đoán.

Mean squared error (MSE)

```
1
2  from sklearn.metrics import mean_squared_error
3  # sklearn có thư viện giúp tính RMSE một cách dễ dàng
4  # tương tự như trên, y_true là vector lưu kết quả chính xác
5  #                    y_pred là vector lưu dự đoán
6  y_true = [3, -0.5, 2, 7]
7  y_pred = [2.5, 0.0, 2, 8]
8  mean_squared_error(y_true, y_pred)
```

Mean Absolute Error (MAE)

Mean Absolute Error (MAE) là độ đo để đánh giá các mô hình hồi quy. MAE được định nghĩa là trung bình tổng trị tuyệt đối sai số giữa đầu ra dự đoán và kết quả thực:

$$MAE = \frac{1}{n}\sum_{1}^{n}|y_i - \hat{y}_i|.$$

MAE có miền giá trị từ $[0, +\infty]$. Trên cùng tập dữ liệu, MAE càng nhỏ thì có độ chính xác càng cao.

Mean Absolute Error (MAE)

```
1
2  from sklearn.metrics import mean_absolute_error #gọi thư viện để tính MAE
3  #giả sử y_true là vector lưu kết quả chính xác
4  #       y_pred là vector lưu dự đoán
5  y_true = [3, -0.5, 2, 7]
6  y_pred = [2.5, 0.0, 2, 8]
7  mean_absolute_error(y_true, y_pred)
```

Nội dung chính

Bias và Variance (Độ lệch và phương sai)

* Năng lực của những mô hình phân loại và dự báo trong lớp các mô hình học có giám sát của machine learning thường được thể hiện qua hai khía cạnh độ chệch (bias) và phương sai (variance).
* Hiểu được chính xác ý nghĩa của hai khái niệm này giúp chúng ta tạo ra những mô hình ít chệch và có độ chính xác đồng đều trên cả tập huấn luyện và tập kiểm tra và đồng thời có khả năng áp dụng mô hình vào thực tiễn mà không lo lắng tới các lỗi phát sinh.

Bias (Độ lệch)

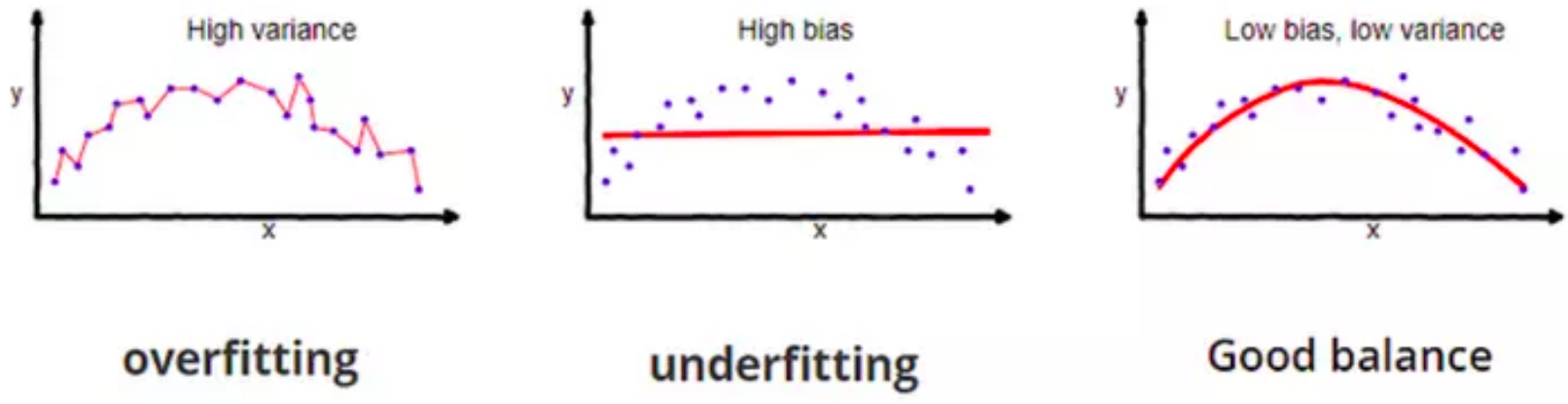
* **Bias**: là sai số giữa giá trị dự đoán trung bình của mô hình và giá trị thực tế. Khi xây dựng mô hình chúng ta mong muốn sẽ tạo ra độ lệch thấp. Tức là giá trị dự báo sẽ gần với ground truth hơn.
* **High bias**: sai số lớn, mô hình đơn giản, tuy nhiên kết quả dự đoán chính xác không cao
* **Low bias**: sai số nhỏ, mô hình phức tạp, kết quả dự đoán tốt

Variance (phương sai)

- **Variance**: là sai số thể hiện mức độ "nhạy cảm" của mô hình với những biến động trong dữ liệu huấn luyện. Mô hình có variance cao thường thể hiện rất tốt trên tập dữ liệu huấn luyện, nhưng không cho kết quả khả quan trên tập dữ liệu kiểm thử.
- **Low-variance**: mô hình ít biến thiên theo sự thay đổi của dữ liệu huấn luyện
- **High-variance**: mô hình biến thiên mạnh, bám sát theo sự thay đổi của dữ liệu huấn luyện.

Nội dung chính

Overfitting và Underfitting (Quá vừa và dưới vừa)



Hiện tượng Underfitting có thể xem như mô hình "học dốt", còn Overfitting cho biết rằng mô hình đang "học vẹt".

Overfitting và Underfitting (Quá vừa và dưới vừa)

* **Underfitting:** là hiện tượng mà mô hình có high bias và low varriance, cho kết quả dự đoán không tốt trên cả tập huấn luyện và tập kiểm thử. Underfitting thường dễ được phát hiện vì cho kết quả tệ trên tập huấn luyện.
* **Overfitting**: là hiện tượng mà mô hình có low bias và high variance, lúc này mô hình trở nên phức tạp, bám sát theo dữ liệu huấn luyện. Mô hình cho kết quả rất tốt trên dữ liệu đã được học, nhưng cho kết quả tệ trên dữ liệu chưa từng gặp bao giờ. Vấn đề này xảy ra khi mô hình cố gắng fit tất cả các điểm dữ liệu huấn luyện, bao gồm cả nhiễu.

Hiện tượng Underfitting có thể xem như mô hình "học dốt", còn Overfitting cho biết rằng mô hình đang "học vẹt".

Nội dung chính

Biến đổi (feature transformation) và trích chọn đặc trưng (feature extraction)

* Biến đổi đặc trưng là những kĩ thuật giúp biến đổi dữ liệu đầu vào thành những dữ liệu phù hợp với mô hình nghiên cứu (Xem Mục 4 Chương 2).
* Không phải toàn bộ thông tin được cung cấp từ một biến dự báo hoàn toàn mang lại giá trị trong việc phân loại. Do đó chúng ta cần phải trích lọc những thông tin chính từ biến đó.

Nội dung chính

Đặc trưng văn bản

* Dữ liệu văn bản có thể đến từ nhiều nguồn và nhiều định dạng khác nhau (kí tự thường, kí tự hoa, kí tự đặc biệt,...). Có nhiều phương pháp xử lý dữ liệu phù hợp với từng đề tài cụ thể. Bài giảng sẽ giới thiệu phương pháp phổ biến nhất.
* Do văn bản là các kí tự nên làm thể nào để lượng hóa được kí tự? Kĩ thuật mã hóa (tokenization) sẽ giúp ta thực hiện điều này. Mã hóa đơn giản là việc chúng ta chia đoạn văn thành các câu văn, các câu văn thành các từ.
* Trong mã hóa thì từ là đơn vị cơ sở. Chúng ta cần một bộ tokenizer có kích thước bằng toàn bộ các từ xuất hiện trong văn bản hoặc bằng toàn bộ các từ có trong từ điển.

Đặc trưng văn bản

* Một câu văn sẽ được biểu diễn bằng một sparse vector mà mỗi một phần tử đại diện cho một từ, giá trị của nó bằng 0 hoặc 1 tương ứng với từ không xuất hiện hoặc có xuất hiện.
* Các bộ tokernizer sẽ khác nhau cho mỗi một ngôn ngữ khác nhau.
* Trong tiếng việt có một bộ tokenizer khá nổi tiếng của nhóm VnCoreNLP nhưng được viết trên ngôn ngữ java. Tốc độ xử lý của java sẽ nhanh hơn trên python đáng kể nhưng mặt hạn chế là phần lớn các data scientist thường không xây dựng model trên java.

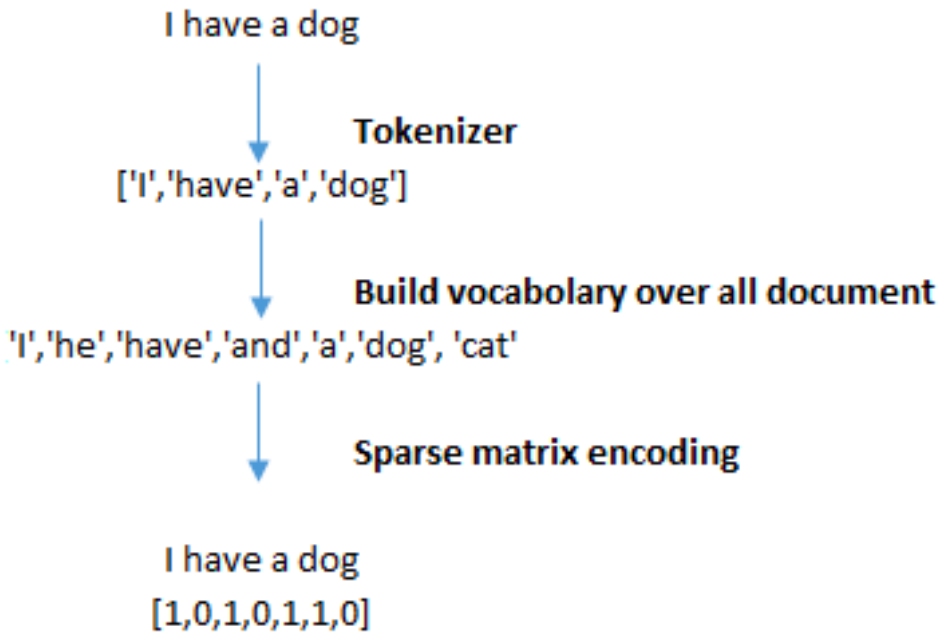
Đặc trưng văn bản

* Một câu văn sẽ được biểu diễn bằng một sparse vector mà mỗi một phần tử đại diện cho một từ, giá trị của nó bằng 0 hoặc 1 tương ứng với từ không xuất hiện hoặc có xuất hiện.
* Các bộ tokernizer sẽ khác nhau cho mỗi một ngôn ngữ khác nhau.
* Trong tiếng việt có một bộ tokenizer khá nổi tiếng của nhóm VnCoreNLP nhưng được viết trên ngôn ngữ java. Tốc độ xử lý của java sẽ nhanh hơn trên python đáng kể nhưng mặt hạn chế là phần lớn các data scientist thường không xây dựng model trên java.
* Sử dụng các túi từ (bags of words) để tạo ra một vector có độ dài bằng độ dài của tokenizer và mỗi phần tử của túi từ sẽ đếm số lần xuất hiện của một từ trong câu và sắp xếp chúng theo một vị trí phù hợp trong vector.

```python
from functools import reduce
import numpy as np

# Đầu vào là một texts bao gồm 3 câu văn:
texts = [['i', 'have', 'a', 'cat'],
         ['he', 'has', 'a', 'dog'],
         ['he', 'has', 'a', 'dog', 'and', 'i', 'have', 'a', 'cat']]

# B1: Xây dựng từ điển
dictionary = list(enumerate(set(reduce(lambda x, y: x + y, texts))))

# B2: Mã hoá câu sang véc tơ tần suất
def bag_of_word(sentence):
    # Khởi tạo một vector có độ dài bằng với từ điển.
    vector = np.zeros(len(dictionary))
    # Đếm các từ trong một câu xuất hiện trong từ điển.
    for i, word in dictionary:
        count = 0
        # Đếm số từ xuất hiện trong một câu.
        for w in sentence:
            if w == word:
                count += 1
        vector[i] = count
    return vector

for i in texts:
    print(bag_of_word(i))
```

Bags of words

Quá trình trên có thể được mô tả bởi biểu đồ dưới đây:



* Ví dụ: "you have no dogs"và "no, you have dogs"là 2 câu văn có biểu diễn giống nhau mặc dù có ý nghĩa trái ngược nhau.

Bags of words

* Với những ứng dụng thực tế, từ điển có thể rất lớn, vì vậy vector đặc trưng thu được sẽ rất dài. Có rất nhiều từ trong từ điển không xuất hiện trong một văn bản. Như vậy các vector đặc trưng thu được thường có rất nhiều phần tử bằng 0 (sparse vector).
* Các biểu diễn theo Bags of words có hạn chế đó là không phân biệt được 2 câu văn có cùng các từ bởi Bags of words không phân biệt thứ tự trước sau của các từ trong một câu.
* Ví dụ: "you have no dogs"và "no, you have dogs"là 2 câu văn có biểu diễn giống nhau mặc dù có ý nghĩa trái ngược nhau.

$\Longrightarrow$ **Phương pháp bag-of-n-gram sẽ được sử dụng để khắc phục.**

Phương pháp bag-of-n-gram

* Phương pháp bag-of-n-grams là phương pháp mở rộng của bag-of-words. Một n-grams là một chuỗi bao gồm n tokens.
* Trong trường hợp $n = 1$ từ ta gọi là unigram, đối với $n = 2$ từ là bigram và $n = 3$ từ là trigram.
* Ví dụ: nếu ta có câu "Tôi yêu học máy", unigram của câu này sẽ là ["Tôi", "yêu", "học", "máy"], còn bigram của câu này sẽ là ["Tôi yêu", "yêu học", "học máy"].
* Như vậy số lượng các từ trong từ điển sẽ gia tăng một cách đáng kể. Nếu chúng ta có k từ đơn thì có thể lên tới k^2 từ trong bigram. Nhưng thực tế không phải hầu hết các từ đều có thể ghép đôi với nhau nên véc tơ biểu diễn của câu trong bigram là một véc tơ rất thưa và có số chiều lớn.

Phương pháp bag-of-n-gram

* Trong sklearn, để sử dụng bigram thì trong CountVectorizer chúng ta thay đổi ngram-range = (2, 2).
* Giá trị đầu tiên là độ dài nhỏ nhất và giá trị sau là độ dài lớn nhất được phép của các ngrams. Ở đây ta khai báo độ dài nhỏ nhất và lớn nhất là 2 nên thu được ngrams là bigram.

```
from sklearn.feature_extraction.text import CountVectorizer

# bigram
bigram = CountVectorizer(ngram_range = (2, 2))
n1, n2, n3 = bigram.fit_transform(['you have no dog', 'no, you have dog', 'you have a dog']).toarray()

# trigram
trigram = CountVectorizer(ngram_range = (3, 3))
n1, n2, n3 = trigram.fit_transform(['you have no dog', 'no, you have dog', 'you have a dog']).toarray()
```

Phương pháp TF-IDF

* Giả sử chúng ta có một bộ văn bản (corpus) bao gồm rất nhiều các văn bản con. Những từ hiếm khi được tìm thấy trong bộ văn bản (corpus) nhưng có mặt trong một số chủ đề nhất định có thể chiếm vai trò quan trọng hơn.
* Ví dụ đối với chủ đề gia đình thì các từ như cha mẹ, ông bà, con cái, anh em, chị em xuất hiện nhiều hơn so với các chủ đề khác.
* Ngoài ra cũng có những từ xuất hiện rất nhiều trong văn bản nhưng chúng xuất hiện ở hầu như mọi chủ đề, mọi văn bản chẳng hạn như the, a, an. Những từ như vậy được gọi là stopwords vì chúng không có nhiều ý nghĩa đối với việc phân loại văn bản.
* Khi mã hoá ngôn ngữ thì chúng ta sẽ tìm cách loại bỏ những từ stopwords bằng cách sử dụng từ điển có sẵn các từ stopwords quan trọng.

Phương pháp TF-IDF

Phương pháp TF-IDF là một phương pháp mà chúng ta sẽ đánh trọng số cho các từ mà xuất hiện ở một vài văn bản cụ thể lớn hơn thông qua công thức:

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D; t \in d\}| + 1} = \log \frac{|D|}{\text{df}(d, t) + 1}$$

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

trong đó:
- $|D|$ là số lượng các văn bản trong bộ văn bản.
- $\text{df}(d, t) = |\{d \in D; t \in d\}|$ là tần suất các văn bản $d \in D$ mà từ t xuất hiện.
- $\text{tf}(t, d)$ là tần suất xuất hiện của từ t trong văn bản d.

Như vậy một từ càng phổ biến khi idf càng nhỏ và tfidf càng lớn.

Phương pháp TF-IDF

Để mã hoá văn bản dựa trên phương pháp tfidf chúng ta sử dụng package sklearn như sau:

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    'tôi thích ăn bánh mì nhân thịt',
    'cô ấy thích ăn bánh mì, còn tôi thích ăn xôi',
    'thị trường chứng khoán giảm làm tôi lo lắng',
    'chứng khoán sẽ phục hồi vào thời gian tới. danh mục của tôi sẽ tăng trở lại',
    'dự báo thời tiết hà nội có mưa vào chiều và tối. tôi sẽ mang ô khi ra ngoài'
]

# Tính tfidf cho mỗi từ. max_df để loại bỏ stopwords xuất hiện ở hơn 90% các câu
vectorizer = TfidfVectorizer(max_df = 0.9)
# Tokenize các câu theo tfidf
X = vectorizer.fit_transform(corpus)
print('words in dictionary:')
print(vectorizer.get_feature_names())
print('X shape: ', X.shape)
```

Phương pháp TF-IDF

```
words in dictionary:
['bánh', 'báo', 'chiều', 'chứng', 'còn', 'có', 'cô', 'của', 'danh', 'dự', 'gian',
'giảm', 'hà', 'hồi', 'khi', 'khoán', 'lo', 'làm', 'lại', 'lắng', 'mang', 'mì',
'mưa', 'mục', 'ngoài', 'nhân', 'nội', 'phục', 'ra', 'sẽ', 'thích', 'thị', 'thịt',
'thời', 'tiết', 'trường', 'trở', 'tăng', 'tối', 'tới', 'và', 'vào', 'xôi', 'ăn',
'ấy']
```

Ta có thể thấy từ tôi xuất hiện ở toàn bộ các câu và không mang nhiều ý nghĩa của chủ đề của câu nên có thể coi là một stopword. Bằng phương pháp lọc cận trên của tần suất xuất hiện từ trong văn bản là 90% ta đã loại bỏ được từ này khỏi dictionary.

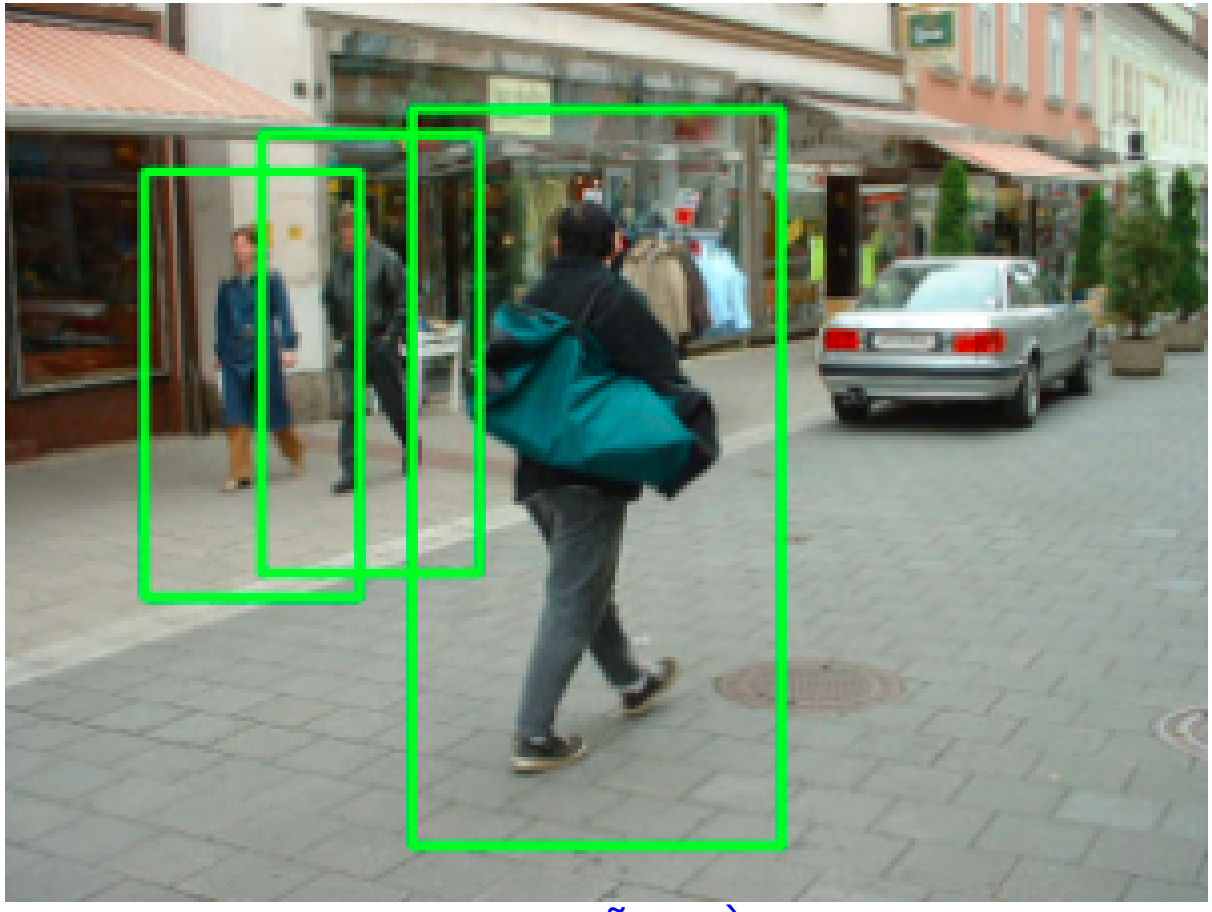
Nội dung chính

Đặc trưng ảnh

* Trước đây, khi tài nguyên tính toán còn hạn chế và thời kì mạng thần kinh vẫn chưa thực sự phát triển, khai phá đặc trưng cho dữ liệu hình ảnh là một lĩnh vực phức tạp. Khi đó cần thiết kế những bộ trích lọc thủ công để trích lọc các đặc trưng như góc, cạnh, đường nét ngang, dọc, chéo, biên, màu sắc, ...
* Trước khi deep learning bùng nổ, thuật toán thường được sử dụng trong xử lý ảnh đó chính là HOG (histogram of oriented gradient).
https://phamdinhkhanh.github.io/2019/11/22/HOG.html

Đặc trưng ảnh

* HOG có những ứng dụng cụ thể như
  * Nhận diện người (human detection)



Đặc trưng ảnh

* Nhận diện khuôn mặt (face detection)

* Nhận diện các vật thể khác
* Tạo feature cho các bài toán phân loại ảnh

Đặc trưng ảnh

Tuy nhiên, hiện nay có rất nhiều các phương pháp khác nhau trong computer vision.

* Khi phân loại ảnh, chúng ta có thể áp dụng họ các mô hình CNN (Inception Net, mobile Net, Resnet, Dense Net, Alexnet, Unet,...).
* Khi phát hiện vật thể là các mô hình YOLO, SSD, Faster RCNN, Fast RCNN, Mask RCNN...

Đặc trưng ảnh

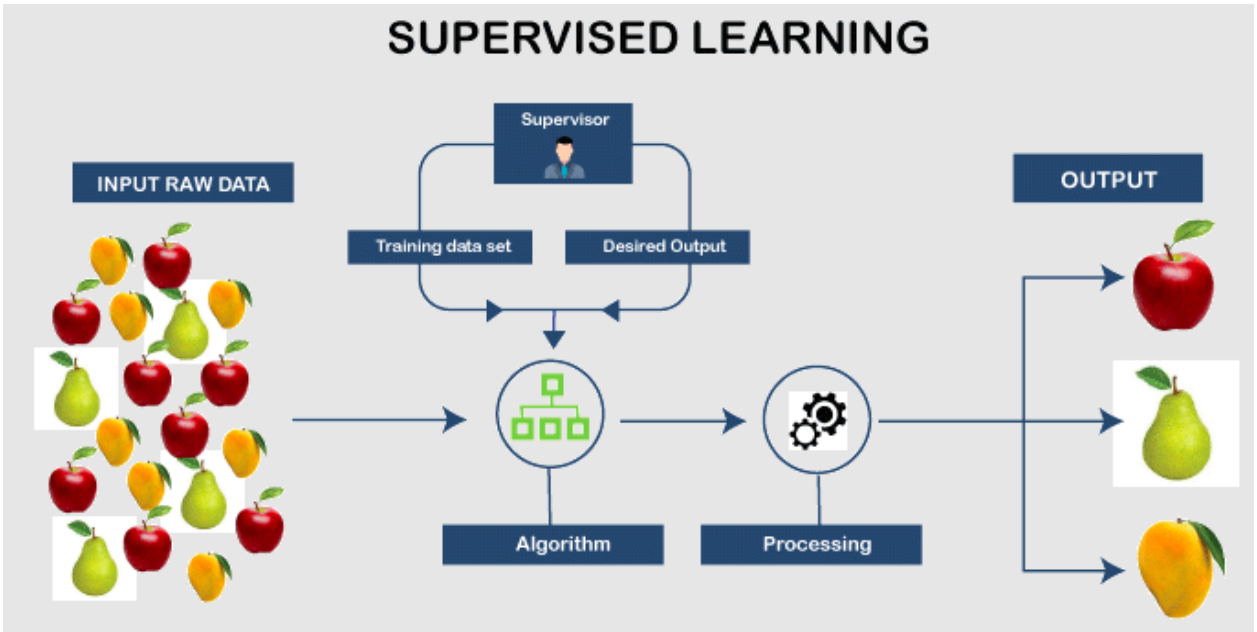
* Những kiến trúc end-to-end cho phép các bộ trích lọc đặc trưng gắn liền với bộ phân loại trong một pipeline duy nhất.
* Nhờ các nguồn tài nguyên gồm các mô hình pretrained sẵn có mà nên không cần phải tìm ra kiến trúc và huấn luyện mạng từ đầu mà có thể tải xuống một mạng hiện đại đã được huấn luyện.
* Điều chỉnh để thích ứng với các mạng này theo nhu cầu bằng cách "tách" các lớp kết nối đầy đủ (fully connected layers) cuối cùng của mạng, thêm các lớp mới được thiết kế cho một nhiệm vụ cụ thể, và sau đó đào tạo mạng trên dữ liệu mới.
* Nếu nhiệm vụ chỉ là vector hóa hình ảnh, thì chỉ cần loại bỏ các lớp cuối cùng và sử dụng kết quả đầu ra từ các lớp trước đó.

Nội dung chính

Nội dung chính

Học có giám sát (Học có giám sát)

* Supervised learning là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (input, outcome) đã biết từ trước. Cặp dữ liệu này còn được gọi là (data, label), tức (dữ liệu, nhãn).



Học có giám sát (Supervised learning)

Cụ thể,

* Tập hợp biến đầu vào $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ và một tập hợp nhãn tương ứng $\mathcal{Y} = \{y_1, y_2, \ldots, y_N\}$, trong đó x_i, y_i là các vector.
* Các cặp dữ liệu biết trước $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ được gọi là tập training data (dữ liệu huấn luyện). Từ tập training data này, chúng ta cần tạo ra một hàm số ánh xạ mỗi phần tử từ tập $\mathcal{X}$ sang một phần tử (xấp xỉ) tương ứng của tập $\mathcal{Y}$:

$$y_i \approx f(x_i), \forall i = 1, 2, \ldots, N$$

* Mục đích là xấp xỉ hàm số f thật tốt để khi có một dữ liệu x mới, chúng ta có thể tính được nhãn tương ứng của nó $y = f(x)$.

Học có giám sát (Supervised learning)

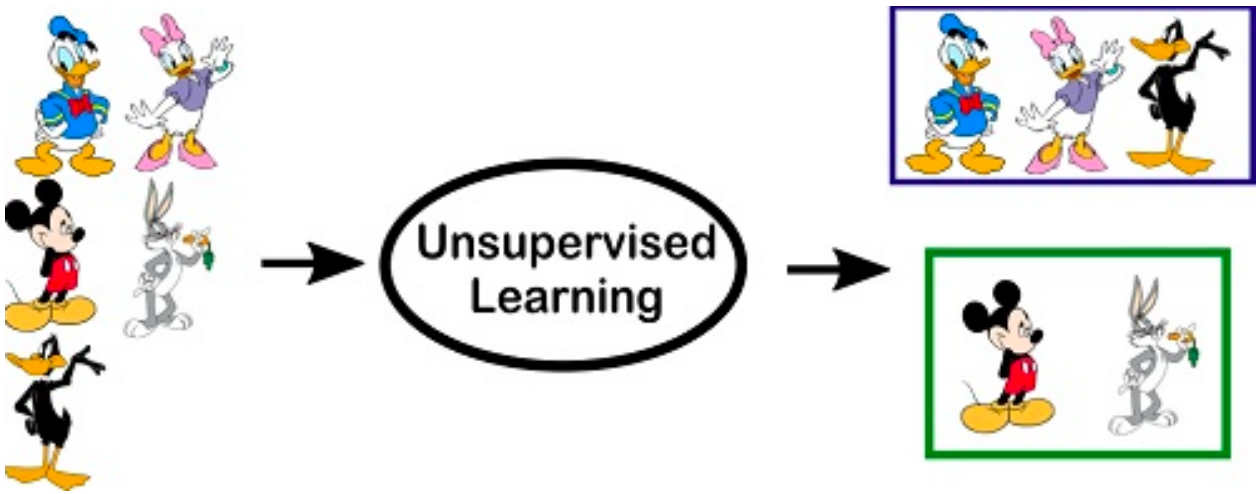
Thuật toán supervised learning còn được tiếp tục chia nhỏ ra thành hai loại chính:

* Classification (Phân loại): Một bài toán được gọi là classification nếu các label của input data được chia thành một số hữu hạn nhóm. Ví dụ: Gmail xác định xem một email có phải là spam hay không; các hãng tín dụng xác định xem một khách hàng có khả năng thanh toán nợ hay không.
* Regression (Hồi quy): Nếu label không được chia thành các nhóm mà là một giá trị thực cụ thể. Ví dụ: một căn nhà rộng x m^2, có y phòng ngủ và cách trung tâm thành phố z km sẽ có giá là bao nhiêu?

Nội dung chính

Học không giám sát (Unsupervised Learning)

* Unsupervised Learning là thuật toán không biết được outcome hay nhãn mà chỉ có dữ liệu đầu vào. Nó sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó. ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimension reduction) để thuận tiện trong việc lưu trữ và tính toán.
* Unsupervised learning là chỉ có dữ liệu vào $\mathcal{X}$ mà không biết nhãn $\mathcal{Y}$ tương ứng.



Học không giám sát (Unsupervised Learning)

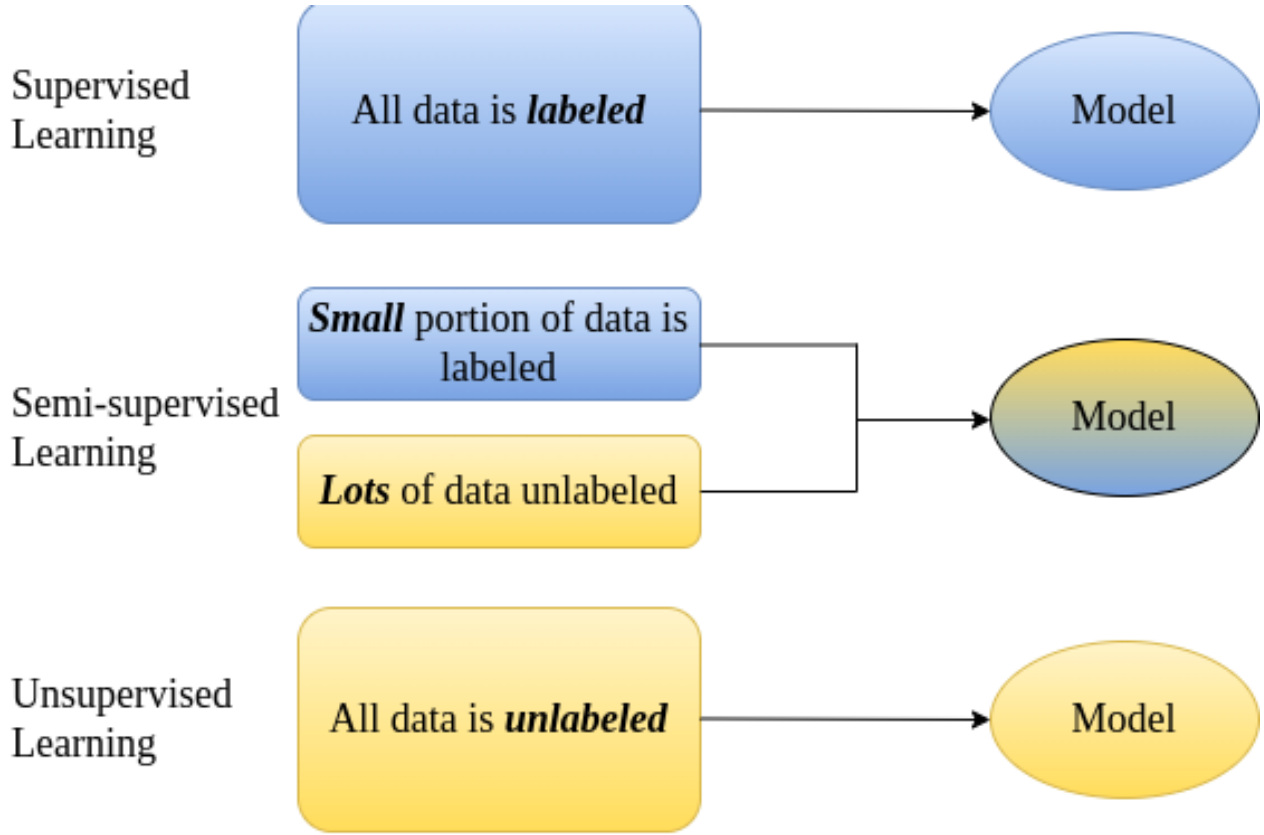
Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:

* Clustering (phân nhóm): Một bài toán phân nhóm toàn bộ dữ liệu thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng.
* Association (liên kết): Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng.

Nội dung chính

Học bán giám sát (Semi-Supervised Learning)

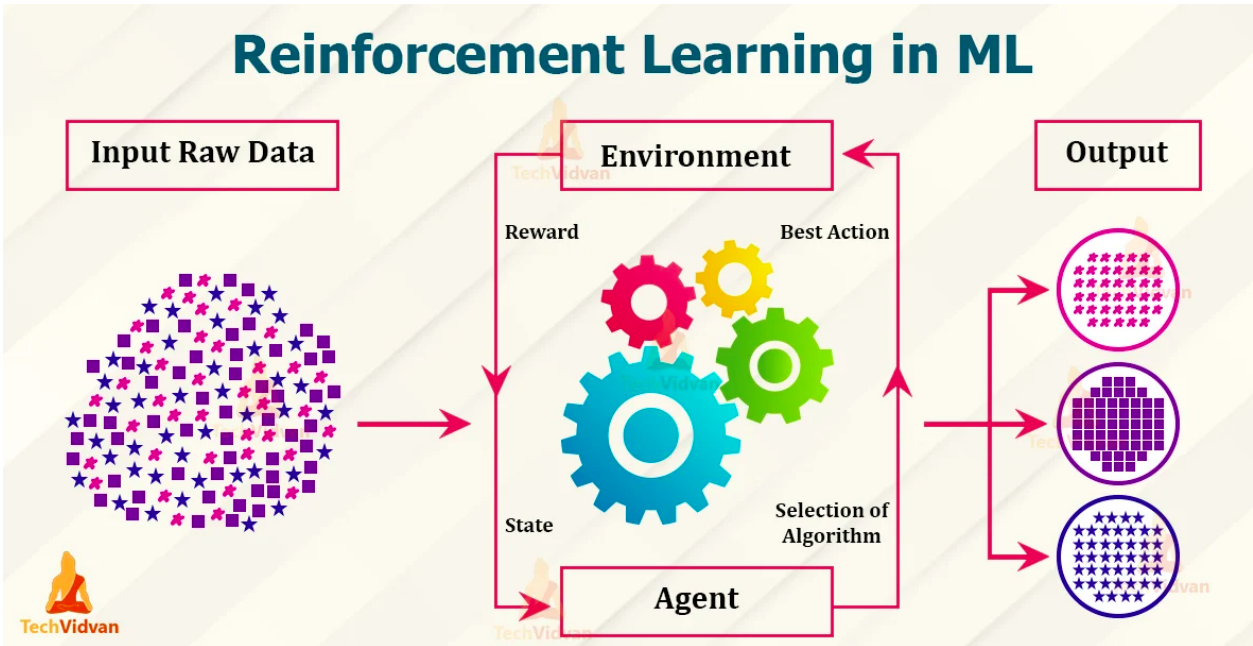
Các bài toán khi có một lượng lớn dữ liệu nhưng chỉ một phần được gán nhãn được gọi là Semi-Supervised Learning, nằm giữa hai nhóm Supervised learning và Unsupervised Learning.



Nội dung chính

Học kết hợp (Reinforcement Learning)

Reinforcement learning giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất. Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.



Bài tập nhóm

Mỗi nhóm tìm hiểu 01 thuật toán Machine Learning sau:

1. Linear Regresion
2. K-means Clustering
3. K-nearest neighbors
4. Perceptron Learning Algorithm
5. Multi-layer Perceptron
6. Logistic Regression
7. Support Vector Machine
8. Soft Margin Support Vector Machine
9. Kernel Support Vector Machine
10. Multi-class Support Vector Machine
11. Softmax Regression
12. Naive Bayes Classifier