

# Bài 1 - Kỹ thuật feature engineering

07 Jan 2019 - phamdinhhkhanh

## Menu

- 1. Giới thiệu về feature engineering
- 2. Trích lọc đặc trưng (feature extraction).
  - 2.1. Trích lọc đặc trưng cho văn bản.
  - 2.2. Trích lọc đặc trưng trong xử lý ảnh
  - 2.3. Thông tin địa lý
  - 2.4. Dữ liệu thời gian
- 3. Biến đổi dữ liệu (feature transformation)
- 4. Lựa chọn feature (feature selection)
  - 4.1. Phương pháp thống kê.
  - 4.2. Sử dụng mô hình
  - 4.3. Sử dụng grid search
- 5. Nhận xét.
- 6. Tài liệu tham khảo.

## 1. Giới thiệu về feature engineering

Hiện nay các phương pháp học máy xuất hiện ngày càng nhiều và trở nên mạnh mẽ hơn. Các mô hình học máy như mạng neural network, Random Forest, Decision Tree, SVM, kNN,... đều là những mô hình có tính tường minh thấp, độ chính xác cao, độ phức tạp và tính linh hoạt cao. Các mô hình học máy đa dạng sẽ làm phong phú thêm sự lựa chọn của các modeler. Tuy nhiên bên cạnh việc áp dụng các phương pháp mạnh, modeler cần phải chuẩn hóa dữ liệu tốt, bởi dữ liệu là nguyên liệu để mô hình dựa trên đó xây dựng một phương pháp học. Nếu mô hình học trên một bộ dữ liệu không tốt, kết quả dự báo sẽ không tốt. Nếu mô hình học trên một bộ dữ liệu tốt, kết quả mô hình tự khắc sẽ được cải thiện. Chính vì thế vai trò của chuẩn hóa dữ liệu quan trọng đến mức Andrew Nguyen đã từng nói 'xây dựng mô hình machine learning không gì khác là thực hiện feature engineering'. Và thực tế cũng cho thấy trong các cuộc thi phân tích dữ liệu, các leader board đều áp dụng tốt các kỹ thuật tạo đặc trưng bên cạnh việc áp dụng những phương pháp mạnh. Những mô hình đơn giản nhưng được xây dựng trên biến chất lượng thường mang lại hiệu quả hơn những mô hình phức tạp như mạng nơ ron hoặc các mô hình kết hợp nhưng được xây dựng trên biến chưa được sử dụng các kỹ thuật tạo đặc trưng.

Về kỹ thuật tạo đặc trưng chúng ta có 3 phương pháp chính:

- **Trích lọc feature:** 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. Chẳng hạn như trong các mô hình chuỗi thời gian chúng ta thường sử dụng kỹ thuật phân rã thời gian để trích lọc ra các đặc trưng như Ngày thành Năm, Tháng, Quý,... Các đặc trưng mới sẽ giúp phát hiện các đặc tính chu kỳ và mùa vụ, những đặc tính mà thường xuất hiện trong các chuỗi thời gian. Kỹ thuật trích lọc đặc trưng thông thường được áp dụng trên một số dạng biến như:
  1. Trích lọc đặc trưng trong xử lý ảnh và xử lý ngôn ngữ tự nhiên: Các mạng nơ ron sẽ trích lọc ra những đặc trưng chính và học từ những đặc trưng này để thực hiện tác vụ phân loại.
  2. Dữ liệu về vị trí địa lý: Từ vị trí địa lý có thể suy ra vùng miền, thành thị, nông thôn, mức thu nhập trung bình, các yếu tố về nhân khẩu,...
  3. Dữ liệu thời gian: Phân rã thời gian thành các thành phần thời gian
- **Biến đổi feature:** Biến đổi dữ liệu gốc thành những dữ liệu phù hợp với mô hình nghiên cứu. Những biến này thường có tương quan cao hơn đối với biến mục tiêu và do đó giúp cải thiện độ chính xác của mô hình. Các phương pháp này bao gồm:
  1. Chuẩn hóa và thay đổi phân phối của dữ liệu thông qua các Kỹ thuật feature scaling như Minmax scaling, Mean normalization, Unit length scaling, Standardization.
  2. Tạo biến tương tác: Trong thống kê các bạn hẳn còn nhớ kiểm định ramsey reset test về mô hình có bỏ sót biến quan trọng? Thông qua việc thêm vào mô hình các biến bậc cao và biến tương tác để tạo ra một mô hình mới và kiểm tra hệ số các biến mới có ý nghĩa thống kê hay không. Ý tưởng của tạo biến tương tác cũng gần như thế. Tức là chúng ta sẽ tạo ra những biến mới là các biến bậc cao và biến tương tác.
  3. Xử lý dữ liệu missing: Có nhiều lý do khiến ta phải xử lý missing data. Một trong những lý do đó là dữ liệu missing cũng mang những thông tin giá trị, do đó nếu thay thế được các missing bằng những giá trị gần đúng sẽ mang lại nhiều thông tin hơn cho mô hình. Bên cạnh đó nhiều mô hình không làm việc được với dữ liệu missing dẫn tới lỗi training. Do đó ta cần giải quyết các biến missing. Đối với biến numeric, các phương pháp đơn giản nhất là thay thế bằng mean, median,... Một số kỹ thuật cao cấp hơn sử dụng phân phối ngẫu nhiên để fill các giá trị missing dựa trên phân phối của các giá trị đã biết hoặc sử dụng phương pháp simulate missing value dựa trên trung bình của các quan sát láng giềng. Đối với dữ liệu category, missing value có thể được giữ nguyên như một class độc lập hoặc gom vào các nhóm khác có đặc tính phân phối trên biến mục tiêu gần giống.
- **Lựa chọn feature:** Phương pháp này được áp dụng trong những trường hợp có rất nhiều dữ liệu mà chúng ta cần lựa chọn ra dữ liệu có ảnh hưởng lớn nhất đến sức mạnh phân loại của mô hình. Các phương pháp có thể áp dụng đó là ranking các biến theo mức độ quan trọng bằng các mô hình như Random Forest, Linear Regression, Neural Network, SVD,...; Sử dụng chỉ số IV trong scorecard; Sử dụng các chỉ số khác như AIC hoặc Pearson Correlation, phương sai. Chúng ta có thể phân chia các phương pháp trên thành 3 nhóm:
  1. Cách tiếp cận theo phương pháp thống kê: Sử dụng tương quan Pearson Correlation, AIC, phương sai, IV.
  2. Lựa chọn đặc trưng bằng sử dụng mô hình: Random Forest, Linear Regression, Neural Network, SVD.

Top

3. Lựa chọn thông qua lưới (grid search): Coi số lượng biến như một thông số của mô hình. Thử nghiệm các kịch bản với những số lượng biến khác nhau. Các bạn có thể xem cách thực hiện grid search (<https://miguelmalvarez.com/2015/02/23/python-and-kaggle-feature-selection-multiple-models-and-grid-search/>).

Để mô phỏng các kĩ thuật này, chúng ta sẽ sử dụng dữ liệu trong cuộc thi của thi Two Sigma Connect: Rental Listing Inquiries Kaggle competition. File train.json là dữ liệu training. Bài toán của chúng ta là cần dự báo mức độ tin nhiệm của một danh sách những người thuê mới. Chúng ta phân loại danh sách thành 3 cấp độ ['low', 'medium', 'high']. Để đánh giá kết quả chúng ta sử dụng hàm trung bình sai số rmse.

```
1 import json
2 import pandas as pd
3
4 with open('../input/train.json', 'r') as iodata:
5     data = json.load(iodata)
6     dataset = pd.DataFrame(data)
7
8 dataset.head()
```

## 2. Trích lọc đặc trưng (feature extraction).

Trong thực tế dữ liệu thường ở dạng thô, đến từ nhiều nguồn khác nhau như văn bản, các phiếu điều tra, các hệ thống lưu trữ, website, app, .... Nền đòi hỏi người xây dựng mô hình phải thu thập và tổng hợp lại các nguồn dữ liệu có liên quan đến đề tài nghiên cứu. Dữ liệu sau đó phải được làm sạch và chuyển thành dạng có cấu trúc (structure data) để tiến hành xây dựng mô hình. Do đó chúng ta sẽ cần đến các kĩ thuật trích lọc đặc trưng để biến dữ liệu từ dạng thô sơ như text, word, các nhãn sang các biến số học có khả năng định lượng. Một trong những kiểu dữ liệu phổ biến áp dụng kĩ thuật trích lọc này là dữ liệu dạng văn bản sẽ được trình bày bên dưới.

### 2.1. Trích lọc đặc trưng cho 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. Tuy nhiên chúng ta sẽ đi vào phương pháp phổ biến nhất.

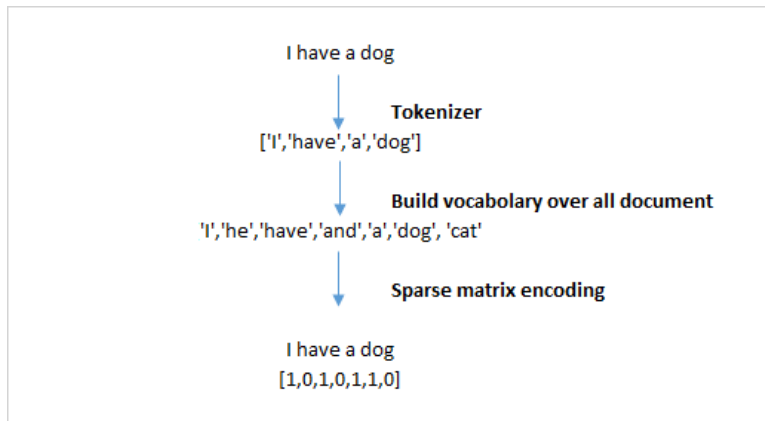
Do văn bản là các kí tự nên làm thế nào để lượng hóa được kí tự? Kĩ thuật mã hóa (tokenization) sẽ giúp ta thực hiện điều này. Mã hóa đơn giản là việc chúng ta chia đoạn văn thành các câu văn, các câu văn thành các từ. Trong mã hóa thì từ là đơn vị cơ sở. Chúng ta cần một bộ tokenizer có kích thước bằng toàn bộ các từ xuất hiện trong văn bản hoặc bằng toàn bộ các từ có trong từ điển. 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 tokenizer sẽ khác nhau cho mỗi một ngôn ngữ khác nhau. Trong tiếng việt có một bộ tokenizer (<https://github.com/vncorenlp/VnCoreNLP>) 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.

Chúng ta 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. Bên dưới là code minh họa cho quá trình này.

```
1 from functools import reduce
2 import numpy as np
3
4 # Giả sử một texts có 3 câu văn là các phần tử trong list như bên dưới
5 texts = [['i', 'have', 'a', 'cat'],
6          ['he', 'have', 'a', 'dog'],
7          ['he', 'and', 'i', 'have', 'a', 'cat', 'and', 'a', 'dog']]
8
9 dictionary = list(enumerate(set(reduce(lambda x, y: x + y, texts))))
10 # Dictionary sẽ chứa toàn bộ các từ của texts.
11
12 def bag_of_word(sentence):
13     # Khởi tạo một vector có độ dài bằng với từ điển.
14     vector = np.zeros(len(dictionary))
15     # Đếm các từ trong một câu xuất hiện trong từ điển.
16     for i, word in dictionary:
17         count = 0
18         # Đếm số từ xuất hiện trong một câu.
19         for w in sentence:
20             if w == word:
21                 count += 1
22         vector[i] = count
23     return vector
24
25 for i in texts:
26     print(bag_of_word(i))
```

Top

Quá trình này có thể được mô tả bởi biểu đồ bên dưới:



Các biểu diễn theo túi từ có hạn chế đó là chúng ta không phân biệt được 2 câu văn có cùng các từ bởi túi từ không phân biệt thứ tự trước sau của các từ trong một câu. Chẳng như 'you have no dog' và 'no, you have dog' là 2 câu văn có biểu diễn giống nhau mặc dù có ý nghĩa trái ngược nhau. Chính vì thế phương pháp N-gram sẽ được sử dụng thay thế.

```

1  from sklearn.feature_extraction.text import CountVectorizer
2
3  vect = CountVectorizer(ngram_range = (1, 1))
4  vect.fit_transform(['you have no dog', 'no, you have dog']).toarray()
  
```

```

1  vect.vocabulary_
  
```

```

1  vect = CountVectorizer(ngram_range = (1, 2))
2  vect.fit_transform(['you have no dog', 'no, you have dog']).toarray()
  
```

```

1  vect.vocabulary_
  
```

```

1  from sklearn.feature_extraction.text import CountVectorizer
2  from scipy.spatial.distance import euclidean
3
4  vect = CountVectorizer(ngram_range = (3, 3), analyzer = 'char_wb')
5  n1, n2, n3, n4 = vect.fit_transform(['andersen', 'peterson', 'petrov', 'smith']).toarray()
6  euclidean(n1, n2), euclidean(n2, n3), euclidean(n3, n4)
  
```

Những từ hiếm khi được tìm thấy trong tập văn bản (corpus) nhưng có mặt trong một văn bản cụ thể có thể quan trọng hơn. Do đó cần tăng trọng số của các nhóm từ ngữ để tách chúng ra khỏi các từ phổ biến. Cách tiếp cận này được gọi là TF-IDF (Term Frequency - Inverse Document Frequency), chúng ta có thể tham khảo tf-idf - wiki (<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>). Các chỉ số chính đánh giá tần suất xuất hiện của một từ trong toàn bộ tập văn bản là idf và tfidf được tính như bên dưới:

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

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

Ở đây

- $|D|$  là số lượng các văn bản trong corpus.
- $df(d, t)$  là số lượng các văn bản mà từ  $t$  xuất hiện.
- $tf(t, d)$  là tần suất các từ xuất hiện trong một văn bản.

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

Để tính tfidf cho các từ trong văn bản ta có thể sử dụng package sklearn như sau:

```

1  from sklearn.feature_extraction.text import TfidfVectorizer
2  corpus = [
3      'tôi thích ăn bánh mì nhân thịt',
4      'cô ấy thích ăn bánh mì, còn tôi thích ăn xôi',
5      'thị trường chứng khoán giảm làm tôi lo lắng',
6      'chứng khoán sẽ phục hồi vào thời gian tới. danh mục của tôi sẽ tăng trở lại',
7      'dự báo thời tiết hà nội có mưa vào chiều và tối. tôi sẽ mang ô khi ra ngoài'
8  ]
9
10 # Khởi tạo model tính tfidf cho mỗi từ
11 # Tham số max_df để loại bỏ các từ stopwords xuất hiện ở hơn 90% các câu.
12 vectorizer = TfidfVectorizer(max_df = 0.9)
13 # Tokenize các câu theo tfidf
14 X = vectorizer.fit_transform(corpus)
15 print('words in dictionary:')
16 print(vectorizer.get_feature_names())
17 print('X shape: ', X.shape)


1  words in dictionary:
2  ['bánh', 'báo', 'chiều', 'chứng', 'còn', 'có', 'cô', 'của', 'danh', 'dự', 'gian', 'giảm', 'hà', 't'
3  X shape: (5, 45)

```

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 stopwords. 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.

Các phương pháp bỏ túi có thể tìm được ở các link như Catch me if you can competition (<https://www.kaggle.com/c/catch-me-if-you-can-intruder-detection-through-webpage-session-tracking>), bag of app (<https://www.kaggle.com/xiaoml/bag-of-app-id-python-2-27392>), bag of event (<http://www.interdigital.com/download/58540a46e3b9659c9f000372>):

Word2Vec là một trường hợp đặc biệt của thuật toán nhúng từ. Sử dụng Word2Vec và các mô hình tương tự, chúng ta không chỉ có thể vector hóa các từ trong một không gian đa chiều (thường là vài trăm chiều) mà còn so sánh sự tương tự về mặt ngữ nghĩa của chúng. Đây là một ví dụ điển hình về các khai triển có thể được thực hiện trên các khái niệm vectorized: king - man + woman = queen

 King - man + woman = queen

Lưu ý rằng mô hình này không hiểu ý nghĩa của các từ mà chỉ cố gắng định vị các vector sao cho các từ được sử dụng trong ngữ cảnh chung gần nhau.

Các mô hình như vậy cần phải được đào tạo trên các tập dữ liệu rất lớn để các tọa độ vector nắm bắt được các ngữ nghĩa. Các mô hình pretrained cho xử lý ngôn ngữ tự nhiên có thể được tải về tại word2vec - api (<https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models>).

Các phương pháp tương tự được áp dụng trong các lĩnh vực khác như trong tin sinh. Một ứng dụng khác nữa là food2vec (<https://jaan.io/food2vec-augmented-cooking-machine-intelligence/>).

## 2.2. Trích lọc đặc trưng trong xử lý ảnh

Xử lý ảnh là một lĩnh vực vừa dễ và vừa khó. Nó dễ bởi chúng ta có thể ứng dụng các mô hình pretrained mà không cần phải suy nghĩ, nhưng nó cũng khó hơn bởi nếu bạn muốn xây dựng một mô hình cho riêng mình đòi hỏi bạn phải thực sự đào sâu vào nó.

Trong quãng thời gian trước đây khi GPU yếu hơn và "thời kì phục hưng của mạng thần kinh" vẫn chưa thực sự xảy ra, các đặc trưng được tạo ra từ hình ảnh là một lĩnh vực phức tạp. Người ta phải làm việc ở các low-level như xác định góc, biên giới của các khu vực, thống kê phân phối màu sắc, v.v. Các chuyên gia có kinh nghiệm về thị giác máy tính có thể đưa ra rất nhiều điểm tương đồng giữa các phương pháp cũ và mạng nơ-ron; đặc biệt là các mạng nơ-ron tích chập hiện đại như Haar cascades ([https://en.wikipedia.org/wiki/Haar-like\\_feature](https://en.wikipedia.org/wiki/Haar-like_feature)). Nếu bạn muốn đọc thêm, dưới đây là một vài liên kết đến một số thư viện thú vị: skimage ([https://en.wikipedia.org/wiki/Haar-like\\_feature](https://en.wikipedia.org/wiki/Haar-like_feature)) và SimpleCV (<http://simplecv.readthedocs.io/en/latest/SimpleCV.Features.html>).

Thông thường trong lĩnh vực computer vision chúng ta sẽ sử dụng mạng nơ-ron tích chập. Bạn không cần phải tìm ra kiến trúc và huấn luyện mạng từ đầu. Thay vào đó, có thể tải xuống một mạng hiện đại đã được pretrained với trọng số từ các nguồn đã được công bố. Các nhà khoa học dữ liệu thường thực hiện điều chỉnh để thích ứng với các mạng này theo nhu cầu của họ 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. Nhiệm vụ của bạn chỉ là vector hóa hình ảnh, bạn chỉ cần loại bỏ các lớp cuối cùng và sử dụng kết quả đầu ra từ các lớp trước đó:

Đây là một mô hình phân lớp được huấn luyện trên một bộ dữ liệu từ trước hay còn gọi là mô hình pretrained. Lớp cuối cùng của mạng được tách ra và sử dụng để huấn luyện lại trên tập dữ liệu mới nhằm điều chỉnh để dự báo cho bộ dữ liệu mới đó.

Top

Tuy nhiên, chúng ta sẽ không tập trung quá nhiều vào kỹ thuật mạng nơ ron. Thay vào đó các feature được tạo thủ công vẫn rất hữu ích: ví dụ đối với bài toán trong cuộc thi Rental Listing Inquiries Kaggle Competition, để dự đoán mức độ phổ biến của danh sách cho thuê, ta có thể giả định rằng các căn hộ có ánh sáng sẽ thu hút nhiều sự chú ý hơn và tạo một feature mới như "giá trị trung bình của pixel". Bạn có thể tìm thấy một số ví dụ thú vị trong tài liệu về PIL (<http://pillow.readthedocs.io/en/3.1.x/reference/ImageStat.html>).

Nếu có văn bản trên hình ảnh, bạn có thể đọc nó để khai thác một số thông tin thông qua gói phát hiện văn bản trong hình ảnh pytesseract (<https://github.com/madmaze/pytesseract>).

```
1  # Cài đặt package pytesseract
2  import sys
3  ![sys.executable] -m pip install pytesseract

1  # Cài đặt tesseract
2  ![sys.executable] -m pip install tesseract

1  from pytesseract import image_to_string
2  from PIL import Image
3  import requests
4  from io import BytesIO
5  import matplotlib.pyplot as plt
6  import numpy as np
7
8  %matplotlib inline
9
10 ##### Just a random picture from search
11 img = 'http://ohscurrent.org/wp-content/uploads/2015/09/domus-01-google.jpg'
12 img = requests.get(img)
13
14 img = Image.open(BytesIO(img.content))
15
16 # show image
17 img_arr = np.array(img)
18 plt.imshow(img_arr)
19
20 # img = Image.open(BytesIO(img.content))
21 # text = image_to_string(img)
22 # text
```

Đọc một hình ảnh thiết kế căn hộ thông qua link.

```
1  img2 = requests.get('https://photos.renthop.com/2/8393298_6acaf11f030217d05f3a5604b9a2f70f.jpg')
2  img2 = Image.open(BytesIO(img2.content))
3  img2 = np.array(img2)
4  plt.imshow(img2)
```



## 2.3. Thông tin địa lý

Trong python chúng ta có một package khá phổ biến trong việc khai thác các thông tin địa lý đó là `reverse_geocoder`. Có 2 dạng bài toán chính với thông tin địa lý gồm geocoding: mã hóa một tọa độ địa lý từ một địa chỉ và revert geocoding: từ thông tin cung cấp về kinh độ và vĩ độ trả về địa chỉ của địa điểm và các thông tin có liên quan. Cả hai bài toán đều có thể giải quyết thông qua API của google map hoặc OpenStreetMap. Sau đây là ví dụ trích xuất thông tin địa lý từ tập dataset của cuộc thi Rental Listing Inquiries Kaggle Competition.

```
1  import sys
2  # install package reverse_geocoder
3  ![sys.executable] -m pip install reverse_geocoder

1  import reverse_geocoder as revgc
2  revgc.search((dataset.latitude[1], dataset.longitude[1]))
```

Như chúng ta thấy, từ tọa độ có thể biết được căn hộ này nằm ở thành phố New York là một nơi phát triển và có mức sống cao. Như vậy mức giá của nó khả năng sẽ cao hơn. Từ quận và huyện ta xác định được căn hộ có nằm ở trung tâm hay không, các tiện nghi xung quanh nó. Những thông tin trên rất quan trọng trong việc đánh giá khả năng bán được của căn hộ. Mặc dù trong bộ dữ liệu gốc không hề xuất hiện nhưng chúng được trích xuất từ tọa độ địa lý.

## 2.4. Dữ liệu thời gian

Top

Trong dự báo, các dữ liệu thường có trạng thái thay đổi. Trạng thái của ngày hôm qua có thể khác biệt so với ngày hôm nay. Chẳng hạn như chiều cao, cân nặng của một người hay giá thị trường của các cổ phiếu. Chính vì thế thời gian là một đại lượng có ảnh hưởng lớn tới biến mục tiêu. Từ một mốc thời gian biết trước chúng ta có thể phân rã nó thành giờ trong ngày, ngày trong tháng, tháng, quý, năm,.... Sẽ có rất nhiều điều thú vị được khám phá từ các thông tin này. Chẳng hạn như qui luật thay đổi theo mùa vụ (nhiệt độ các tháng thay đổi theo mùa, GDP thay đổi theo qui luật quý, ...). Yếu tố thời gian còn giúp xác định xu hướng biến đổi của một đại lượng như thông qua việc theo dõi nhiệt độ của trái đất qua các năm chúng ta biết được trái đất đang nóng lên hay lượng băng tan phản ánh mực nước biển đang dâng lên qua từng năm. Tốc độ thay đổi của một đại lượng kết hợp với tính mùa vụ là một chỉ số quan trọng để ước lượng chuỗi thời gian.

Biến đổi one-hot coding là một phương pháp quan trọng được sử dụng để mã hóa các biến chu kỳ thời gian. One-hot coding sẽ biến đổi một biến thành các vector có phần tử là 0 hoặc 1, trong đó 1 đại diện cho sự xuất hiện của đặc trưng và 0 đại diện cho các đặc trưng mà biến không có. Ví dụ cụ thể: Chúng ta có 1 ngày trong tuần có thể rơi vào các thứ từ 2 đến chủ nhật. Như vậy một biểu diễn one-hot coding của một ngày theo thứ tự ngày trong tuần sẽ là một vector mà phần tử tương ứng với thứ tự trong tuần của ngày đó sẽ bằng 1 và các phần tử còn lại bằng 0. Các biểu diễn này tương tự với mã hóa dữ liệu văn bản thành các sparse vector. Trong python chúng ta có thể sử dụng hàm weekday() để xác định thứ tự của một ngày trong tuần. Thuộc tính weekday() chỉ tồn tại đối với dữ liệu dạng datetime. Do đó ta cần chuyển đổi các biến ngày đang ở dạng string về dạng datetime thông qua strftime (string format time). Bảng string format time có thể xem tại đây (<http://strftime.org/>).

```
1 # dataset['created'].apply(lambda x: x.date().weekday())
2 from datetime import datetime
3
4 def parser(x):
5     # Để biết được định dạng strftime của một chuỗi kí tự ta phải tra trong bảng string format time
6     return datetime.strptime(x, '%Y-%m-%d %H:%M:%S')
7
8
9 dataset['created'] = dataset['created'].map(lambda x: parser(x))
10 #Kiểm tra định dạng time
11 for i, k in zip(dataset.columns, dataset.dtypes):
12     print('{}: {}'.format(i, k))
```

Như vậy biến created đã được chuyển về dạng datetime. Chúng ta có thể tạo ra một one-hot coding dựa vào hàm weekday().

```
1 dataset['weekday'] = dataset['created'].apply(lambda x: x.date().weekday())
2 print(dataset['weekday'].head())
```

Ta có thể tạo ra một biến trả về trạng thái ngày có phải là cuối tuần bằng kiểm tra weekday() có rơi vào [5, 6] hay không.

```
1 dataset['is_weekend'] = dataset['created'].apply(lambda x: 1 if x.date().weekday() in [5, 6] else 0)
2 print(dataset['is_weekend'][:5])
```

Trong bài toán này, một số giao dịch có thể phụ thuộc vào thời gian. Chẳng hạn như lịch trả nợ của thẻ tín dụng sẽ rơi vào kì sao kê là một ngày cụ thể trong tháng. Khi làm việc với dữ liệu chuỗi thời gian chúng ta nên lưu ý tới danh sách các ngày đặc biệt trong năm như nghỉ tết âm lịch, quốc khánh, quốc tế lao động,.... Bởi những ngày này thường sẽ gây ra bất thường về dữ liệu kinh doanh.

#### Hỏi đáp?

H: Liệu các ngày nghỉ tết nguyên đán, quốc tế lao động, quốc khánh,... có đặc điểm gì chung không?

T: Chúng ta nên đưa những ngày này vào danh sách các ngày tiềm năng có bất thường dữ liệu.

#### Chuỗi thời gian từ website, log,...

Các hệ thống website lớn sẽ tracking lại các session của người dùng. Những thông tin được tracking bao gồm thông tin thiết bị, loại event, customer ID, ... Từ customer ID chúng ta có thể link tới database người dùng để biết được các thông tin về giới tính, độ tuổi, tài khoản, hành vi giao dịch,.... Trong một số trường hợp một khách hàng có thể thay đổi thiết bị truy cập, do đó không phải hầu hết các trường hợp chúng ta đều map được session với Customer ID trên dữ liệu local. Tuy nhiên từ các thông tin được lưu trong Cookie về người dùng (còn gọi là user agent) cũng cung cấp cho chúng ta khá nhiều điều. Chẳng hạn như: Thiết bị truy cập, trình duyệt, hệ điều hành,... Từ thiết bị di động chúng ta cũng ước đoán được người dùng có mức thu nhập như thế nào: Sử dụng Iphone X thì khả năng cao là người có thu nhập cao, sử dụng điện thoại xiaomi khả năng là người thu nhập trung bình và thấp,.... Để phân loại các thông tin về người dùng chúng ta có thể sử dụng package user\_agents trong python.

```
1 # Download package user_agents
2 !{sys.executable} -m pip install user_agents
```

Top

```

1  import user_agents
2  # Giả định có một user agent như bên dưới
3  ua = 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/56.0
4  # Parser thông tin user agent
5  ua = user_agents.parse(ua)
6  # Khai thác các thuộc tính của user
7  print('Is a bot? ', ua.is_bot)
8  print('Is mobile? ', ua.is_mobile)
9  print('Is PC? ', ua.is_pc)
10 print('OS Family: ', ua.os.family)
11 print('OS Version: ', ua.os.version)
12 print('Browser Family: ', ua.browser.family)
13 print('Browser Version: ', ua.browser.version)

```

### 3. Biến đổi dữ liệu (feature transformation)

Các chiều dữ liệu thường có sự khác biệt về đơn vị (scale), phân phối (distribution) và điều đó tác động không nhỏ lên hiệu quả phân loại của mô hình và khả năng hội tụ của các thuật toán trượt gradient. Một tập hợp dữ liệu có đơn vị quá khác biệt giữa các biến thường khiến gradient không hội tụ tới cực trị toàn cục. Các khác biệt về đơn vị cũng khiến việc đánh giá ảnh hưởng của các biến bị sai lệch nhiều hơn. Một biến có ảnh hưởng lớn (thể hiện qua độ lớn của hệ số ước lượng) nhưng có độ rộng miền giá trị nhỏ sẽ rất nhạy tới biến mục tiêu bởi chỉ một thay đổi rất nhỏ trong giá trị của nó cũng làm biến mục tiêu thay đổi lớn. Những biến có độ rộng miền giá trị lớn thường có tham số nhỏ, vì vậy chỉ một thay đổi nhỏ trong tham số cũng làm hàm mất mát thay đổi lớn. Điều này dẫn đến thuật toán trượt ngẫu nhiên cần điều chỉnh tốc độ học (learning rate) phù hợp theo độ lớn đơn vị của từng biến. Thiết kế một thuật toán gradient như vậy rất phức tạp. Một cách đơn giản hơn đó là chúng ta sẽ đồng hóa sự khác biệt về mặt đơn vị bằng các kĩ thuật biến đổi dữ liệu. Một số kĩ thuật chính bao gồm:

1. Biến đổi Standardization. Biến được rescaling theo kì vọng và độ lệch chuẩn như sau:

$$\mathbf{x} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma(\mathbf{x})}$$

1. Rescaling theo range. Biến được đưa về các range [0,1] hoặc [-1,1] mà vẫn giữ nguyên tỷ lệ khoảng cách giữa các biến. Chẳng hạn khi các đặc trưng nằm trong khoảng [0,1]:

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}$$

Khi đặc trưng nằm khoảng [-1, 1] công thức sẽ như sau:

$$\mathbf{x}' = \frac{2(\mathbf{x} - \bar{\mathbf{x}})}{\max(\mathbf{x}) - \min(\mathbf{x})}$$

1. Rescaling về unit length. Khi đó độ dài khoảng cách norm2 bằng 1 (tức trong không gian Euclid vecor sẽ có độ lớn bằng 1).

$$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$$

Dữ liệu được biến đổi theo phương pháp 1 sẽ có phân phối chuẩn hóa với kì vọng bằng 0 và phương sai bằng 1. Chính vì thế nên cách 1 còn được gọi là biến đổi dữ liệu theo phân phối chuẩn (Z-score distribution hoặc normal distribution). Dữ liệu được biến đổi theo cách 2 sẽ giới hạn độ lớn của các biến trong miền giá trị [0, 1]. Theo phương pháp 3 các biến sẽ có tổng bình phương các giá trị bằng 1. So với các cách biến đổi 1 và 2 thì giá trị của biến ở phương pháp 3 sẽ nhỏ hơn rất nhiều. Chính vì vậy khi lựa chọn biến đổi theo phương pháp 3 chúng ta nên điều chỉnh learning rate nhỏ hơn một số lần so với phương pháp 1 và 2.

Tất cả các biến đổi dữ liệu đều được thực hiện thông qua sklearn.preprocessing. Đây là một package rất cơ bản và đầy đủ về các mô hình machine learning truyền thống. Bao gồm các chức năng: xây dựng mô hình hồi qui, tiền xử lý dữ liệu,....

Bên dưới sẽ là thực hành biến đổi dữ liệu theo:

#### Phân phối chuẩn:

Scale theo phân phối chuẩn được thực hiện thông qua hàm StandardScaler.

```

1  from sklearn.preprocessing import StandardScaler
2  from scipy.stats import beta
3  from scipy.stats import shapiro
4  import statsmodels.api as sm
5  import numpy as np
6
7  # Tạo ra một chuỗi phân phối beta
8  data = beta(1, 10).rvs(1000).reshape(-1, 1)
9  print('data shape:%s'%str(data.shape))
10 # Sử dụng kiểm định shapiro để kiểm tra tính phân phối chuẩn.
11 shapiro(data)

```

Top

```

1  # Giá trị tới hạn và p-value
2  shapiro(StandardScaler().fit_transform(data))
3
4  # Giá trị tới hạn > p-value chúng ta sẽ bác bỏ giả thuyết về phân phối chuẩn.

```

Scale theo phân phối chuẩn đối với biến prices trong bộ dữ liệu dataset.

```

1  # biến đổi dữ liệu theo phân phối chuẩn:
2  price = np.float64(dataset.price.values)
3  print('Head 5 of original prices:', price[:5])
4  price_std = StandardScaler().fit_transform(price.reshape(-1, 1))
5  print('Head 5 of standard scaling prices:\n', price_std[:5])

1  # Biến đổi trên tương đương với công thức sau:
2  price_std = (price - price.mean()) / price.std()
3  print('Head 5 of standard scaling prices:\n', price_std[:5])

```

### Theo range minmax

Chúng ta có thể sử dụng MinMaxScaler để biến đổi dữ liệu.

```

1  from sklearn.preprocessing import MinMaxScaler
2  price_mm = MinMaxScaler().fit_transform(price.reshape(-1, 1))
3  print('Head of min max scaling price:\n', price_mm[:5])

1  # Hoặc đơn giản hơn là
2  price_mm = (price - price.min()) / (price.max() - price.min())
3  print('Head of min max scaling price:\n', price_mm[:5])

```

Chúng ta có thể kiểm tra tính phân phối chuẩn của các biến sau scale dựa trên biểu đồ Q-Q plot biểu diễn giá trị phân phối thực tế dựa trên giá trị phân phối lý thuyết (các giá trị được tính ra từ phân phối lý thuyết, ở đây là phân phối chuẩn). Nếu đường biểu diễn nằm sát đường chéo chính thì biến có khả năng cao đạt tính phân phối chuẩn.

```

1  # Tạo biến log price
2  price_log = np.log(price)
3  # Kiểm tra tính phân phối của các biến price, price_std, price_mm, price_log dựa trên biểu đồ Q-Q
4  sm.qqplot(price, loc = price.mean(), scale = price.std())

```

```

1  sm.qqplot(price_mm, loc = price_mm.mean(), scale = price_mm.std())

```

```

1  sm.qqplot(price_std, loc = price_std.mean(), scale = price_std.std())

```

```

1  sm.qqplot(price_log, loc = price_log.mean(), scale = price_log.std())

```

Trong các giá trị này ta có thể nhận thấy biến có khả năng phân phối chuẩn cao nhất chính là biến price\_log. Trong biểu đồ qqplot ta nhận thấy các điểm nằm rời rạc ở phần đầu và đuôi của đồ thị là những dữ liệu outlier. Nếu ta loại bỏ những dữ liệu này sẽ tạo ra một đồ thị có tính phân phối chuẩn cao hơn. Thật vậy:

```

1  price_rm_outlier = price_log[(price_log < 12) & (price_log > 6)]
2  sm.qqplot(price_rm_outlier, loc = price_rm_outlier.mean(), scale = price_rm_outlier.std())

```

## 4. Lựa chọn feature (feature selection)

Để xây dựng mô hình chúng ta sẽ rất cần đến thông tin. Tuy nhiên số lượng thông tin bao nhiêu là đủ để có được một mô hình mạnh? Ngày nay với sự bùng nổ của bigdata, dữ liệu dường như trở nên quá tải khiến chúng ta không thể đưa toàn bộ dữ liệu vào mô hình. Vì vậy modeler cần phải lựa chọn các biến quan trọng để hồi qui thay vì lựa chọn nhiều biến nhất có thể. Bởi những hạn chế của việc có quá nhiều features đó là:

1. Tăng chi phí tính toán.
2. Quá nhiều biến giải thích có thể dẫn tới overfitting. Tức hiện tượng mô hình hoạt động rất tốt trên tập train nhưng kém trên tập test.
3. Trong số các biến sẽ có những biến gây nhiễu và làm giảm chất lượng mô hình.
4. Rối loạn thông tin do không thể kiểm soát và hiểu hết các biến.

Để khắc phục những hạn chế đó chúng ta sẽ cần đến những thuật toán quan trọng được sử dụng để lựa chọn các biến như sau:



## 4.1. Phương pháp thống kê.

Một trong những phương pháp quan trọng trong các phương pháp thống kê nhằm giảm số lượng biến là lựa chọn dựa trên phương sai. Các biến không biến động sẽ có khả năng giải thích kém hơn. Do đó ý tưởng chính của phương pháp này là tính phương sai của toàn bộ các biến numeric và loại bỏ biến nếu nó nhỏ hơn một ngưỡng threshold.

```
1 from sklearn.feature_selection import VarianceThreshold
2 from sklearn.datasets import make_classification
3
4 # Lấy dữ liệu example từ package sklearn
5 X, y = make_classification()
6
7 print('X: \n', X[:5, :5])
8 print('y: \n', y)
9 print('X shape:', X.shape)
10 print('y shape:', y.shape)
```

Nếu coi 0.5 là ngưỡng threshold của phương sai.

```
1 VarianceThreshold(.5).fit_transform(X).shape
```

Tập dữ liệu vẫn còn nguyên các biến do toàn bộ các phương sai đều > 0.5. Nếu ta nâng ngưỡng này lên 0.9.

```
1 VarianceThreshold(0.9).fit_transform(X).shape
```

Khi đó dữ liệu chỉ còn 16 biến có mức biến động lớn. Ngoài phương pháp phương sai, chúng ta có thể áp dụng phương pháp thống kê dựa trên các chỉ số như  $\chi^2$ ,  $F$  của phân phối chi bình phương và Fisher. Các phương pháp này sẽ đo lường sức mạnh của mô hình khi loại bỏ lần lượt các biến và tìm ra  $k$  biến tốt nhất. Chi tiết về phương pháp thống kê ([https://scikit-learn.org/stable/modules/feature\\_selection.html#univariate-feature-selection](https://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection)) cho bạn đọc quan tâm. Chúng ta sẽ áp dụng cả 2 phương pháp đo lường phương sai và phương pháp thống kê để đánh giá hiệu quả mô hình trước và sau lựa chọn biến.

```
1 from sklearn.feature_selection import SelectKBest, f_classif
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.model_selection import cross_val_score
4
5 X_kbest = SelectKBest(f_classif, k = 5).fit_transform(X, y)
6 X_kvar = VarianceThreshold(0.9).fit_transform(X)
7 print('X shape after applying statistical selection: ', X_kbest.shape)
8 print('X shape after apply variance selection: ', X_kvar.shape)
```

Chúng ta sẽ cùng đánh giá hiệu quả mô hình bằng cross validation trước và sau lựa chọn biến với KFold = 5. Lưu ý quan trọng là khi so sánh hiệu quả các mô hình khác nhau trên một tập dữ liệu chúng ta luôn phải sử dụng cross validation thay vì chia dữ liệu ngẫu nhiên và chỉ so sánh kết quả dựa trên một lần chia mẫu. Bởi cross validation sẽ cover hết toàn bộ tập dữ liệu và mô hình hồi qui được xây dựng trên một tập số lớn dữ liệu (n-1 folds trên tổng số n folds) nên sẽ sát với mô hình hồi qui từ toàn bộ dữ liệu.

```
1 # Hồi qui logistic
2 logit = LogisticRegression(solver='lbfgs', random_state=1)
3
4 # Cross validation cho:
5 # 1. dữ liệu gốc
6 acc_org = cross_val_score(logit, X, y, scoring = 'accuracy', cv = 5).mean()
7 # 2. Áp dụng phương sai
8 acc_var = cross_val_score(logit, X_kvar, y, scoring = 'accuracy', cv = 5).mean()
9 # 3. Áp dụng phương pháp thống kê
10 acc_stat = cross_val_score(logit, X_kbest, y, scoring = 'accuracy', cv = 5).mean()
11
12 print('Accuracy trên dữ liệu gốc:', acc_org)
13 print('Accuracy áp dụng phương sai:', acc_var)
14 print('Accuracy dụng pp thống kê:', acc_stat)
```

Như vậy ta thấy sau khi áp dụng feature selection đã cải thiện được độ chính xác của mô hình dự báo.

## 4.2. Sử dụng mô hình

Đây là phương pháp rất thường xuyên được áp dụng trong các cuộc thi phân tích dữ liệu. Chúng ta sẽ dựa trên một số mô hình cơ sở để đánh giá mức độ quan trọng của các biến. Có 2 lớp mô hình thường được sử dụng để đánh giá biến đó là Random Forest hoặc Linear Regression. Ưu điểm của các phương pháp này là kết quả đánh giá rất chuẩn xác, tuy nhiên nhược điểm của chúng là phải xây dựng mô hình hồi qui rồi mới xác định được biến quan trọng. Điều này dường như đi trái lại với thực tế phải lựa chọn biến trước khi training mô hình. Để áp dụng phương pháp này chúng ta thực hiện như sau:

Top

```

1  from sklearn.ensemble import RandomForestClassifier
2  from sklearn.svm import LinearSVC
3  from sklearn.feature_selection import SelectFromModel
4  from sklearn.model_selection import cross_val_score
5  from sklearn.pipeline import make_pipeline
6
7  # Hồi qui theo RandomForest
8  rdFrt = RandomForestClassifier(n_estimators = 10, random_state = 1)
9  # Hồi qui theo LinearSVC
10 lnSVC = LinearSVC(C=0.01, penalty="l1", dual=False)
11 # Tạo một pipeline thực hiện lựa chọn biến từ RandomForest model và hồi qui theo logit
12 pipe1 = make_pipeline(StandardScaler(), SelectFromModel(estimator = rdFrt), logit)
13 # Tạo một pipeline thực hiện lựa chọn biến từ Linear SVC model và hồi qui theo logit
14 pipe2 = make_pipeline(StandardScaler(), SelectFromModel(estimator = lnSVC), logit)
15 # Cross validate đối với
16 # 1. Mô hình logit
17 acc_log = cross_val_score(logit, X, y, scoring = 'accuracy', cv = 5).mean()
18 # 2. Mô hình RandomForest
19 acc_rdf = cross_val_score(rdFrt, X, y, scoring = 'accuracy', cv = 5).mean()
20 # 3. Mô hình pipe1
21 acc_pip1 = cross_val_score(pipe1, X, y, scoring = 'accuracy', cv = 5).mean()
22 # 3. Mô hình pipe2
23 acc_pip2 = cross_val_score(pipe2, X, y, scoring = 'accuracy', cv = 5).mean()
24
25 print('Accuracy theo logit:', acc_log)
26 print('Accuracy theo random forest:', acc_rdf)
27 print('Accuracy theo pipeline 1:', acc_pip1)
28 print('Accuracy theo pipeline 2:', acc_pip2)

```

Như vậy select dựa trên mô hình Random Forest và Linear SVC đã có hiệu quả trong việc cải thiện độ chính xác của mô hình. Bên cạnh việc thực hiện lựa chọn biến dựa trên model, chúng ta còn có thể lựa chọn biến theo grid search.

## 4.3. Sử dụng grid search

Đây là phương pháp có thể coi là đáng tin cậy nhất trong việc lựa chọn biến quan trọng. Ý tưởng chính của phương pháp này đó là huấn luyện mô hình trên một tập dữ liệu con, lưu lại kết quả sau train, lặp lại quá trình huấn luyện trên những mẫu con khác, so sánh chất lượng các mô hình dự báo để tìm ra một tập các biến tốt nhất. Phương pháp này còn được gọi là Exhaustive Feature Selection ([http://rasbt.github.io/mlxtend/user\\_guide/feature\\_selection/ExhaustiveFeatureSelector/](http://rasbt.github.io/mlxtend/user_guide/feature_selection/ExhaustiveFeatureSelector/)).

Nếu như chúng ta tìm kiếm trên toàn bộ các bộ kết hợp tham số của mô hình sẽ rất lâu. Do đó việc đầu tiên ta cần thực hiện là giới hạn không gian search space. Ban đầu ta cố định trước một số lượng biến N, đi qua lần lượt các kết hợp của toàn bộ N biến đó và lựa chọn ra bộ kết hợp tốt nhất. Khi xét với N+1 biến thì ta sẽ cố định bộ kết hợp tốt nhất của N biến trước đó và chỉ thêm 1 biến mới vào bộ kết hợp này. Quá trình này tiếp tục cho đến khi số lượng các biến đạt mức tối đa hoặc tới khi hàm loss function mô hình không giảm nữa. Phương pháp này gọi là Sequential Feature Selection ([http://rasbt.github.io/mlxtend/user\\_guide/feature\\_selection/SequentialFeatureSelector/](http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/)).

Ý tưởng của Sequential Feature Selection có thể thực hiện ngược lại theo cách cố định một tập hợp lớn nhất các biến, sau đó loại lần lượt các biến cho đến khi các phẩm chất của mô hình không còn được cải thiện. Khi đó bộ kết hợp các biến tối ưu sẽ được lựa chọn. Bên dưới ta sẽ tiến hành sử dụng phương pháp lựa chọn grid search đối với Sequential Feature Selection.

```

1  !{sys.executable} -m pip install mlxtend
2  from mlxtend.feature_selection import SequentialFeatureSelector
3
4  selector = SequentialFeatureSelector(logit, scoring = 'accuracy',
5                                     verbose = 2,
6                                     k_features = 3,
7                                     forward = False,
8                                     n_jobs = -1)
9
10 selector.fit(X, y)

```

Ta có thể thấy mô hình xuất phát từ 20 biến ban đầu và sau mỗi một quá trình sẽ loại dần các biến cho đến khi số lượng biến tối thiểu đạt được là 3 được khai báo trong hàm SequentialFeatureSelector. Sau mỗi quá trình mức độ accuracy sẽ tăng dần.

## 5. Nhận xét.

Như vậy sau bài này các bạn đã nhận ra được Feature Engineering quan trọng như thế nào trong việc tạo ra một mô hình dự báo có sức mạnh. Tổng hợp lại các phương pháp feature engineering:

1. Trích lọc feature: Ứng dụng trong deep learning như xử lý ảnh và xử lý ngôn ngữ tự nhiên, phân rã thời gian, làm việc với dữ liệu địa lý, dữ liệu người dùng tracking từ các hệ thống web, app.
2. Biến đổi feature: Minmax scaling, Unit length scaling, Standardization.
3. Lựa chọn feature: Sử dụng phương pháp thống kê, mô hình hoặc grid search.

Top

Câu hỏi đặt ra:

Bên cạnh những thuật toán, modeler có cần kiến thức về lĩnh vực nghiên cứu (knowledge domain) không?

Để xây dựng một mô hình tốt không chỉ cần có Kiến thức về mô hình mà các hiểu biết về lĩnh vực cũng rất quan trọng. Khi hiểu rõ lĩnh vực, modeler nắm rõ bản chất mối quan hệ của các biến không chỉ qua các con số mà còn trên thực tiễn. Đó cũng là lý do trong một dự án phân tích dữ liệu luôn cần những BA tư vấn nghiệp vụ để giúp modeler hiểu sâu hơn về mối quan hệ trên thực tiễn.

Trong mọi mô hình có nên thực hiện Feature Engineering?

Hầu hết các mô hình hiện đại đều thực hiện feature engineering trước khi training mô hình bởi sau khi thực hiện feature engineering chúng ta sẽ có cơ hội tạo ra một mô hình mạnh hơn. Cần so sánh nhiều mô hình khác nhau để lựa chọn ra đâu là mô hình phù hợp nhất, trong một số trường hợp có thể sử dụng kết hợp các mô hình.

Ý tưởng về Feature Engineering rất nhiều? Làm thế nào để tìm ra một Feature Engineering tối ưu?

Không có câu trả lời cụ thể cho một phương pháp Feature Engineering nào là tối ưu. Chỉ có quá trình thử nghiệm mới rút ra được phương pháp Feature Engineering nào sẽ phù hợp với bài toán cụ thể nào.

## 6. Tài liệu tham khảo.

1. Giới thiệu về feature engineering - mlcourse.ai  
([https://mlcourse.ai/notebooks/blob/master/jupyter\\_english/topic06\\_features\\_regression/topic6\\_feature\\_engineering\\_feature\\_selection.ipynb?flush\\_cache=true](https://mlcourse.ai/notebooks/blob/master/jupyter_english/topic06_features_regression/topic6_feature_engineering_feature_selection.ipynb?flush_cache=true))
2. feature engineering - blog machinelearningcoban - Vu Huu Tiep  
(<https://machinelearningcoban.com/general/2017/02/06/featureengineering/>)
3. tfidf - Information retrieval - wiki (<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>)
4. package pytesseract - ứng dụng trong OCR - blog pyimagesearch (<https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>)
5. extract time in python - blog hamelg (<http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-17.html>)
6. feature scaling - rpub phamdinhhkhanh (<http://rpubs.com/phamdinhhkhanh/398690>)
7. feature scaling - arsenyinfo (<https://www.kaggle.com/arsenyinfo/easy-feature-selection-pipeline-0-55-at-lb>)

Top