

# SESSION 1

## 1. Linear Regression

1.1. Formula

1.2. Large Data

1.2.1. SGD

1.3. Implement LR

1.3.1. Data

1.3.2. Cross-validation

1.3.3. Implement

## 2. Preprocessing Text Data

2.1. Bag of words & TF-IDF

2.1.1. TF-IDF

2.2. Word2Vec

2.2.1. Reference

2.2.2. Word2Vec Model

## 1. Linear Regression

### 1.1. Formula

❑ Lỗi trên tập dữ liệu D:

$$\text{RSS}(f) / N = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2$$

❑ Phương pháp tối thiểu hổ L:

$$w^* = (X^T X)^{-1} X Y \quad \text{với } X = \begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1K} \\ 1 & x_{21} & x_{22} \dots x_{2K} \\ \dots & \dots & \dots \\ 1 & x_{N1} & x_{N2} \dots x_{NK} \end{pmatrix}, Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}$$

❑ Ridge Regression: thêm vào đại lượng phạt  $\lambda \|w\|_2^2$  vào RSS(f)

$$L = \sum_{i=1}^N (y_i - f(x_i))^2 + \lambda \sum_{j=0}^K w_j^2$$

❑ Minimize L ta có  $w^*$  lúc này là<sup>[1]</sup>:

$$w^* = (X^T X + \lambda I_{K+1})^{-1} X Y \quad \text{với } I_{K+1} \text{ là ma trận đơn vị}$$

## 1.2. Large Data

Matrix algebra

Operation	Input	Complexity
Matrix multiplication	Two nxn matrices	$O(n^{2.373})$
Matrix multiplication	One nxm matrix & one m xp matrix	$O(nmp)$
Matrix inversion	One nxn matrix	$O(n^3)$ $O(n^{2.807})$

❑ Khi kích thước dữ liệu quá lớn việc tính nghịch đảo ma trận quá tốn kém. Các giải quyết là tối ưu sử dụng stochastic gradient.

### 1.2.1. SGD

- ❑ Các lược đồ tối ưu dựa vào gradient:

$$w = w - \text{learning\_rate} * \nabla_w L$$

- ❑  $\nabla_w L = E_q[B(w)]$ , lấy mẫu ngẫu nhiên data từ phân phối q và gọi  $b(w)$  là gradient trên tập mẫu:

$$w = w - \text{learning\_rate} * b(w) . b là lấy mẫu độc lập từ B.$$

Việc tối ưu theo stochastic gradient đảm bảo tính hội tụ và về mặt thực nghiệm cho kết quả tốt hơn trên các hàm non-convex so với gradient thông thường.

- ❑ Hiểu đơn giản là ta chia dữ liệu thành các minibatch rồi tối ưu parameter theo gradient của minibatch đó. Lặp lại trên dữ liệu nhiều epoch.

#### ❑ Lược đồ tối ưu:

Linear Regression:  $w = w - \text{learning\_rate} * x^T(xw - y)$

Ridge Regression:  $w = w - \text{learning\_rate} * [x^T(xw - y) + \lambda w]$

## 1.3. Implement LR

### 1.3.1. Data

<https://people.sc.fsu.edu/~jb Burkardt/datasets/regression/x28.txt>

### 1.3.2. Cross-validation

- ❑ Cross-validation (k-fold cross-validation) dùng để lựa chọn tham số cho mô hình (với ridge regression, đó là giá trị **LAMBDA  $\lambda$** ).

#### ❑ Áp dụng 5-fold cross-validation vào việc lựa chọn LAMBDA

❑ 5-fold cross-validation được tiến hành như sau:

➤ Chia  $D_{train}$  thành 5 phần (xấp xỉ) bằng nhau:  $D_1, D_2, D_3, D_4, D_5$

➤ Với mỗi  $D_i$  ( $i = \overline{1, 5}$ ), ta thực hiện:

\* Huấn luyện mô hình trên  $D_{train} \setminus D_i$

\* Tính lỗi trên  $D_i$

➤ Tính lỗi trung bình qua 5 lần

➤ Lựa chọn LAMBDA đem lại lỗi trung bình nhỏ nhất.

➤ Huấn luyện mô hình trên toàn bộ  $D_{train}$  với LAMBDA tìm được và đánh giá hiệu quả mô hình trên  $D_{test}$

---

#### 8.2.2 Cross-validation

Trong nhiều trường hợp, chúng ta có rất hạn chế số lượng dữ liệu để xây dựng mô hình. Nếu lấy quá nhiều dữ liệu trong training set ra làm dữ liệu validation, phần dữ liệu còn lại của training set là không đủ để xây dựng mô hình. Lúc này, validation set phải thật nhỏ để giữ được lượng dữ liệu cho training đủ lớn. Tuy nhiên, một vấn đề khác nảy sinh. Khi validation set quá nhỏ, hiện tượng overfitting lại có thể xảy ra với training set còn lại. Có giải pháp nào cho tình huống này không?

Câu trả lời là *cross-validation*.

*Cross-validation* là một cải tiến của validation với lượng dữ liệu trong validation set là nhỏ nhưng chất lượng mô hình được đánh giá trên nhiều tập validation khác nhau. Một cách thường được sử dụng là chia training set ra  $k$  tập con không giao nhau, có kích thước gần bằng nhau. Tại mỗi lần, được gọi là một *run*, một trong số  $k$  tập con được lấy ra làm validation set. Nhiều mô hình khác nhau sẽ được xây dựng dựa vào hợp của  $k - 1$  tập con còn lại. Mô hình cuối được xác định dựa trên trung bình của các training error và validation error. Cách làm này còn có tên gọi là **k-fold cross-validation**.

### 8.3 Regularization

Một nhược điểm lớn của *cross-validation* là số lượng mô hình cần huấn luyện tỉ lệ thuận với  $k$ . Điều đáng nói là mô hình polynomial regression như trên chỉ có một tham số liên quan đến độ phức tạp của mô hình cần xác định là bậc của đa thức. Trong các nhiều bài toán, lượng tham số cần xác định thường lớn hơn nhiều, và khoảng giá trị của mỗi tham số cũng rộng hơn nhiều, chưa kể đến việc có những tham số có thể là số thực. Điều này dẫn đến việc huấn luyện nhiều mô hình là khó khả thi. Có một kỹ thuật giúp số mô hình cần huấn luyện giảm đi nhiều, thậm chí chỉ một mô hình. Kỹ thuật này có tên gọi là *regularization*.

*Regularization*, một cách dễ hiểu, là thay đổi mô hình một chút, chấp nhận hy sinh độ chính xác trong training set, nhưng giảm độ phức tạp của mô hình, giúp tránh overfitting trong khi vẫn giữ được tính tổng quát của nó. Dưới đây là một vài kỹ thuật regularization.

### 8.3.2 Thêm số hạng vào hàm mất mát

Kỹ thuật regularization phổ biến hơn là thêm vào hàm mất mát một số hạng nữa. Số hạng này thường dùng để đánh giá độ phức tạp của mô hình với giá trị lớn thể hiện mô hình phức tạp. *Hàm mất mát mới* này được gọi là **regularized loss function**, thường được định nghĩa như sau:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda R(\theta)$$

Nhắc lại rằng  $\theta$  được dùng để ký hiệu các tham số trong mô hình.  $\mathcal{L}(\theta)$  là hàm mất mát phụ thuộc vào training set và  $\theta$ ,  $R(\theta)$  là số hạng regularization chỉ phụ thuộc vào  $\theta$ . Số vô hướng  $\lambda$  thường là một số dương nhỏ, còn được gọi là *tham số regularization (regularization parameter)*. Tham số regularization thường được chọn là các giá trị nhỏ để đảm bảo nghiệm của bài toán tối ưu  $\mathcal{L}_{\text{reg}}(\theta)$  không quá xa nghiệm của bài toán tối ưu  $\mathcal{L}(\theta)$ .

Hai hàm regularization phổ biến là  $\ell_1$  norm và  $\ell_2$  norm regularization (viết gọn là  $\ell_1$  regularization và  $\ell_2$  regularization). Ví dụ, khi chọn  $R(\mathbf{w}) = \|\mathbf{w}\|_2^2$  cho hàm mất mát của linear regression, chúng ta sẽ đạt được ridge regression. Hàm regularization này khiến các hệ số trong  $\mathbf{w}$  không quá lớn, giúp tránh việc đầu ra phụ thuộc quá nhiều vào một đặc trưng nào đó. Trong khi đó, khi chọn  $R(\mathbf{w}) = \|\mathbf{w}\|_1$ , nghiệm  $\mathbf{w}$  tìm được có xu hướng rất nhiều phần tử bằng không (*sparse solution*<sup>2</sup>). Khi thêm  $\ell_1$  regularization vào hàm mất mát của linear regression, chúng ta sẽ thu được LASSO regression. Các thành phần khác không của  $\mathbf{w}$  tương đương với các đặc trưng quan trọng đóng góp vào việc dự đoán đầu ra. Các đặc trưng ứng với thành phần bằng không của  $\mathbf{w}$  được coi là ít quan trọng. Chính vì vậy, LASSO regression cũng được coi là một phương pháp giúp lựa chọn những đặc trưng hữu ích cho mô hình; nó cũng là một trong các phương pháp *feature selection*.

$\ell_1$  regularization được cho là giúp cho mô hình *robust* (ít bị ảnh hưởng bởi nhiễu) hơn so với  $\ell_2$  regularization. Tuy nhiên, hạn chế của  $\ell_1$  regularization là đạo hàm của  $\ell_1$  norm không xác định tại không (đạo hàm của hàm trị tuyệt đối), dẫn đến việc tìm nghiệm thường tốn thời gian hơn. Trong khi đó, đạo hàm của  $\ell_2$  norm xác định mọi nơi, và trong nhiều trường hợp, ta có thể tìm được công thức nghiệm cho phương trình đạo hàm của (regularized) loss function bằng không. Các nghiệm có công thức xác định được gọi là *closed-form solution*.

Trong neural network, việc sử dụng  $\ell_2$  regularization còn được gọi là *weight decay* [KH92]. Ngoài ra, gần đây một phương pháp regularization rất hiệu quả cho neural network được sử dụng là *dropout* [SHK<sup>+</sup>14].

## Assessment model

<https://www.youtube.com/watch?v=ujDDwR2GZZM&t=2673s>

<https://www.youtube.com/watch?v=tvt-2nOAYH4>

## 1.3.3. Implement

- **Problem**

□ Triển khai thuật toán Ridge Regression (trường hợp tổng quát của Linear Regression)

- > Đọc dữ liệu
- > Chuẩn hóa dữ liệu
- > Xây dựng mô hình

□ Lựa chọn **LAMBDA** theo phương pháp cross-validation

- **Step**

- **Read Data**

□ Đọc dữ liệu:

- > Đọc file
- > Chia nội dung thành từng dòng
- > Chia mỗi dòng thành các features
- > X: features từ A1 → A15
- > Y: feature cuối cùng, B

I	A1	A2	A3	...	A13	A14	A15	B
1	36	27	71	...	15	59	59	921.870
2	35	23	72	...	10	39	57	997.875
3	44	29	74	...	6	33	54	962.354
4	47	45	79	...	8	24	56	982.291
5	43	35	77	...	38	206	55	1071.289
6	53	45	80	...	32	72	54	1030.380
7	43	30	74	...	32	62	56	934.700
8	45	30	73	...	4	4	56	899.529

- **Normalize Data**

□ Chuẩn hóa dữ liệu:

- > Các features có miền giá trị lệch nhau
- > Chuẩn hóa để đưa về 1 miền chung
- > Có nhiều phương pháp<sup>[1]</sup>, ta chọn

“Feature Scaling”:

I	A1	A2	A3	...	A13	A14	A15	B
1	36	27	71	...	15	59	59	921.870
2	35	23	72	...	10	39	57	997.875
3	44	29	74	...	6	33	54	962.354
4	47	45	79	...	8	24	56	982.291
5	43	35	77	...	38	206	55	1071.289
6	53	45	80	...	32	72	54	1030.380
7	43	30	74	...	32	62	56	934.700
8	45	30	73	...	4	4	56	899.529

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad \text{với } X' \in [0,1]^{N \times 15}, N \text{ là số điểm dữ liệu}$$

□ Chuẩn hóa dữ liệu:

> Nhắc lại công thức

=> Ta cần thêm feature  $x_{i0} = 1$

vào mỗi điểm dữ liệu

$$w^* = (X^T X + \lambda I_{K+1})^{-1} X Y$$

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1K} \\ 1 & x_{21} & x_{22} \dots x_{2K} \\ \dots \\ 1 & x_{N1} & x_{N2} \dots x_{NK} \end{pmatrix}$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}$$

□ Chuẩn hóa dữ liệu:

```

25 def normalize_and_add_ones(X):
26     X = np.array(X)
27     X_max = np.array([[np.amax(X[:, column_id])
28                         for column_id in range(X.shape[1])],
29                         for _ in range(X.shape[0])])
30     X_min = np.array([[np.amin(X[:, column_id])
31                         for column_id in range(X.shape[1])],
32                         for _ in range(X.shape[0])])
33
34     X_normalized = (X - X_min) / (X_max - X_min)
35
36     ones = np.array([[1] for _ in range(X_normalized.shape[0])])
37     return np.column_stack((ones, X_normalized))

```

I	A1	A2	A3	A13	A14	A15	B	
1	36	27	71	...	15	59	59	921.870
2	35	23	72	...	10	39	57	997.875
3	44	29	74	...	6	33	54	962.354
4	47	45	79	...	8	24	56	982.291
5	43	35	77	...	38	206	55	1071.289
6	53	45	80	...	32	72	54	1036.380
7	43	30	74	...	32	62	56	934.700
8	45	30	73	...	4	4	56	899.529

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

○ Implement

□ Triển khai mô hình:

> Xây dựng lớp RidgeRegression

```

40 class RidgeRegression:
41     def __init__(self):
42         return
43
44     def fit(self, X_train, Y_train, LAMBDA):...
53
54     def predict(self, W, X_new):...
58
59     def compute_RSS(self, Y_new, Y_predicted):...
63
64     def get_the_best_LAMBDA(self, X_train, Y_train):...

```

$$w^* = (X^T X + \lambda I_{K+1})^{-1} X Y$$

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1K} \\ 1 & x_{21} & x_{22} \dots x_{2K} \\ \dots \\ 1 & x_{N1} & x_{N2} \dots x_{NK} \end{pmatrix}$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}$$

## □ Triển khai mô hình: > Hàm fit\_gradient

```

13     def fit_gradient_descent(self,X_train,Y_train,LAMBDA,learning_rate,max_num_epoch=100,batch_size=128):
14         W = np.random.randn(X_train.shape[1])
15         last_loss=10e+8
16         for ep in range(max_num_epoch):
17             arr = np.array(range(X_train.shape[0]))
18             np.random.shuffle(arr)
19             X_train=X_train[arr]
20             Y_train=Y_train[arr]
21             total_minibatch = int(np.ceil(X_train.shape[0]/batch_size))
22             for i in range(total_minibatch):
23                 index = i *batch_size
24                 X_train_sub = X_train[index:index+batch_size]
25                 Y_train_sub = Y_train[index:index+batch_size]
26                 grad = X_train_sub.T.dot(X_train_sub.dot(W) -Y_train_sub) + LAMBDA * W
27                 W = W - learning_rate*grad
28             new_loss = self.compute_RSS(self.predict(W,X_train),Y_train)
29             if(np.abs(new_loss - last_loss) <= 1e-5):
30                 break
31             last_loss=new_loss
32     return W

```

## □ Triển khai mô hình:

### > Hàm predict

```

54     def predict(self, W, X_new):
55         X_new = np.array(X_new)
56         Y_new = X_new.dot(W)
57     return Y_new

```

$$Y_{\text{new}} = X_{\text{new}}W$$

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} \dots x_{1K} \\ 1 & x_{21} & x_{22} \dots x_{2K} \\ \dots \\ 1 & x_{N1} & x_{N2} \dots x_{NK} \end{pmatrix}$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}, W = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_K \end{pmatrix}$$

## □ Triển khai mô hình:

### > Xác định giá trị LAMBDA tốt nhất:

- \* B1: Xác định miền giá trị tìm kiếm
- \* B2: Thực hiện cross-validation với từng giá trị LAMBDA có thể
- \* B3: Xác định giá trị LAMBDA tốt nhất trong miền
- \* B4: Quay trở lại bước 1

## □ Triển khai mô hình: Hàm get\_the\_best\_LAMBDA

```

64     def get_the_best_LAMBDA(self, X_train, Y_train):
65         def cross_validation(num_folds, LAMBDA):...
66
67         def range_scan(best_LAMBDA, minimum_RSS, LAMBDA_values):...
68
69         best_LAMBDA, minimum_RSS = range_scan(best_LAMBDA=0, minimum_RSS=10000 ** 2,
70                                              LAMBDA_values=range(50)) # [0, 1, 2, ..., 49]
71
72         LAMBDA_values = [k * 1. / 1000 for k in range(
73             max(0, (best_LAMBDA - 1) * 1000), (best_LAMBDA + 1) * 1000, 1)
74             ] # step size = 0.001
75
76         best_LAMBDA, minimum_RSS = range_scan(best_LAMBDA=best_LAMBDA, minimum_RSS=minimum_RSS,
77                                              LAMBDA_values=LAMBDA_values)
78
79     return best_LAMBDA

```

#### ❑ Triển khai mô hình: Hàm `cross_validation`

```
65 def cross_validation(num_folds, LAMBDA):
66     row_ids = np.array(range(X_train.shape[0]))
67     # np.split() requires equal divisions
68     valid_ids = np.split(row_ids[:len(row_ids) - len(row_ids) % num_folds], num_folds)
69     valid_ids[-1] = np.append(valid_ids[-1], row_ids[len(row_ids) - len(row_ids) % num_folds:])
70     train_ids = [[k for k in row_ids if k not in valid_ids[i]] for i in range(num_folds)]
71     aver_RSS = 0
72     for i in range(num_folds):
73         valid_part = {'X': X_train[valid_ids[i]], 'Y': Y_train[valid_ids[i]]}
74         train_part = {'X': X_train[train_ids[i]], 'Y': Y_train[train_ids[i]]}
75         W = self.fit(train_part['X'], train_part['Y'], LAMBDA)
76         Y_predicted = self.predict(W, valid_part['X'])
77         aver_RSS += self.compute_RSS(valid_part['Y'], Y_predicted)
78     return aver_RSS / num_folds
```

#### ❑ Triển khai mô hình: Chạy thử

```
100 if __name__ == '__main__':
101     X, Y = get_data(path='..//datasets/death-rates-data.txt')
102     # normalization
103     X = normalize_and_add_ones(X)
104     X_train, Y_train = X[:50], Y[:50]
105     X_test, Y_test = X[50:], Y[50:]
106
107     ridge_regression = RidgeRegression()
108     best_LAMBDA = ridge_regression.get_the_best_LAMBDA(X_train, Y_train)
109     print 'Best LAMBDA:', best_LAMBDA
110     W_learned = ridge_regression.fit(
111         X_train=X_train, Y_train=Y_train, LAMBDA=best_LAMBDA
112     )
113     Y_predicted = ridge_regression.predict(W=W_learned, X_new=X_test)
114
115     print ridge_regression.compute_RSS(Y_new=Y_test, Y_predicted=Y_predicted)
```

## 2. Preprocessing Text Data

### 2.1. Bag of words & TF-IDF

#### Biểu diễn Bag of words và TF-IDF cho doc

##### ❑ TF-IDF = term frequency-inverse document frequency

##### ❑ Được sử dụng cho dữ liệu dạng văn bản (text)

##### ❑ Biểu diễn TF-IDF đối với 1 văn bản d trong một tập văn bản (corpus) D:

$$r_d = [\text{tf-idf}(w_1, d, D), \text{tf-idf}(w_2, d, D), \dots, \text{tf-idf}(w_{|V|}, d, D)]$$

với  $r_d \in \mathbb{R}^{|V|}$  là 1 vector  $|V|$  chiều

$V = \{w_i\}$  là từ điển (tập hợp các từ xuất hiện trong D) đối với D

- ❑ Trong đó, mỗi giá trị  $tf\text{-}idf(w_i, d, D)$  được tính như sau:

$$tf\text{-}idf(w_i, d, D) = tf(w_i, d) \times idf(w_i, D)$$

với  $tf(w_i, d) = \frac{f(w_i, d)}{\max\{f(w_j, d) : w_j \in V\}}$

$$idf(w_i, D) = \log_{10} \frac{|D|}{|\{d' \in D : w_i \in d'\}|}$$

- ❑ Trong đó,  $f(w_i, d)$  là số lần xuất hiện của từ  $w_i$  trong văn bản  $d$ .

- ❑ Xác định từ điển  $V$ :

➤ Với mỗi văn bản  $d$  trong  $D$ :

\* B1: Tách  $d$  thành các từ theo punctuations<sup>[1]</sup> ta thu được  $W_d$ :

'Data-Science Lab;2018' -> ['Data', 'Science', 'Lab', '2018']

\* B2: Loại bỏ từ dừng (stop words<sup>[2]</sup>) khỏi  $W_d$ :

$$W_d = W_d \setminus \{\text{stop\_words}\}$$

a, an, the, have, for, ....

\* B3: Đưa các từ về dạng gốc (stemming<sup>[3]</sup>):

$$W_d = \{\text{stem}(w) : w \in W_d\}$$

trong đó  $\text{stem}(w)$  là dạng gốc của  $w$

- ❑ Một cách xác định từ điển  $V$ :

➤ Với mỗi văn bản  $d$  trong  $D$ : thu được  $W_d$

➤ Cuối cùng, ta có:

$$V = \bigcup_{d \in D} W_d$$

### 2.1.1. TF-IDF

Calculate TF-IDF for all text in file data

tf-idf - Wikipedia

In information retrieval, tf-idf (also TF\*IDF, TFIDF, TF-IDF, or Tf-idf), short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling.

W <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

## Regular expression syntax cheatsheet - JavaScript | MDN

This page provides an overall cheat sheet of all the capabilities of RegExp syntax by aggregating the content of the articles in the RegExp guide. If you need more information on a specific topic, please

 [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_Expressions/Cheatsheet](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet)

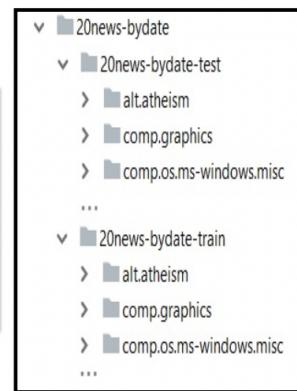


### 1. Read Data & Gather it

#### ❑ Tiền xử lý: đọc dữ liệu và tập hợp dữ liệu

##### > Lấy danh sách các thư mục và newsgroups

```
87 def gather_20newsgroups_data():
88     path = '../datasets/20news-bydate/'
89     dirs = [path + dir_name + '/'
90             for dir_name in listdir(path)
91             if not isfile(path + dir_name)]
92     train_dir, test_dir = (dirs[0], dirs[1]) if 'train' in dirs[0] \
93     else (dirs[1], dirs[0])
94     list_newsgroups = [newsgroup
95                         for newsgroup in listdir(train_dir)]
96     list_newsgroups.sort()
```



##### > Thu thập dữ liệu

```
98 with open('../datasets/20news-bydate/stop_words.txt') as f:
99     stop_words = f.read().splitlines()
100 from nltk.stem.porter import PorterStemmer
101 stemmer = PorterStemmer()
102
103 def collect_data_from(parent_dir, newsgroup_list):...
128
129 train_data = collect_data_from(
130     parent_dir=train_dir,
131     newsgroup_list=list_newsgroups
132 )
133 test_data = collect_data_from(
134     parent_dir=test_dir,
135     newsgroup_list=list_newsgroups
136 )
```

##### > Thu thập dữ liệu: hàm collect\_data\_from

```
103 def collect_data_from(parent_dir, newsgroup_list):
104     data = []
105     for group_id, newsgroup in enumerate(newsgroup_list):
106         label = group_id
107         dir_path = parent_dir + '/' + newsgroup + '/'
108         files = [(filename, dir_path + filename)
109                  for filename in listdir(dir_path)
110                  if isfile(dir_path + filename)]
111         files.sort()
```

### > Thu thập dữ liệu: hàm collect\_data\_from

```
112     for filename, filepath in files:
113         with open(filepath) as f:
114             text = f.read().lower()
115             # remove stop words then stem remaining words
116             words = [stemmer.stem(word)
117                     for word in re.split('\W+', text)
118                     if word not in stop_words]
119             # combine remaining words
120             content = ' '.join(words)
121             assert len(content.splitlines()) == 1
122             data.append(str(label) + '<fff>' +
123                         filename + '<fff>' + content)
124
return data
```

### > Ghi ra file:

```
135 full_data = train_data + test_data
136 with open('../datasets/20news-bydate/20news-train-processed.txt', 'w') as f:
137     f.write('\n'.join(train_data))
138
139 with open('../datasets/20news-bydate/20news-test-processed.txt', 'w') as f:
140     f.write('\n'.join(test_data))
141
142 with open('../datasets/20news-bydate/20news-full-processed.txt', 'w') as f:
143     f.write('\n'.join(full_data))
```

### > Output: 20news-train-processed.txt

```
1 0<fff>49960<fff>mathew mathew manti co uk subject alt atheism faq at
2 0<fff>51060<fff>mathew mathew manti co uk subject alt atheism faq in
3 0<fff>51119<fff>i3150101 dbstul rz tu bs de benedikt rosenau subject
4 0<fff>51120<fff>mathew mathew manti co uk subject re univers violat
5 0<fff>51121<fff>strom watson ibm com rob strom subject re soc motss
6 0<fff>51122<fff>i3150101 dbstul rz tu bs de benedikt rosenau subject
7 0<fff>51123<fff>keith cco caltech edu keith allan schneider subject
8 0<fff>51124<fff>i3150101 dbstul rz tu bs de benedikt rosenau subject
9 0<fff>51125<fff>keith cco caltech edu keith allan schneider subject
10 0<fff>51126<fff>keith cco caltech edu keith allan schneider subject
```

## 2. Preprocessing

### ❑ Tiền xử lý: tạo từ điển và tính trước giá trị idf

```
61 def generate_vocabulary(data_path):
62     def compute_idf(df, corpus_size):...
63
64     with open(data_path) as f:
65         lines = f.read().splitlines()
66         doc_count = defaultdict(int)
67         corpus_size = len(lines)
68
69     for line in lines:
70         features = line.split('<fff>')
71         text = features[-1]
72         words = list(set(text.split()))
73         for word in words:
74             doc_count[word] += 1
```

### ❑ Tiền xử lý: tạo từ điển và tính trước giá trị idf

```
78 words_idfs = [(word, compute_idf(document_freq, corpus_size))
79     for word, document_freq in
80         zip(doc_count.keys(), doc_count.values())
81     if document_freq > 10 and not word.isdigit()]
82 words_idfs.sort(key=lambda (word, idf): -idf)
83 print 'Vocabulary size: {}'.format(len(words_idfs))
84 with open('../datasets/20news-bydate/words_idfs.txt', 'w') as f:
85     f.write('\n'.join([word + '<fff>' + str(idf) for word, idf in words_idfs]))
```

### ❑ Tiền xử lý: tạo từ điển và tính trước giá trị idf

```
62 def compute_idf(df, corpus_size):
63     assert df > 0
64     return np.log10(corpus_size * 1. / df)
```

$$\text{idf}(w_i, D) = \log_{10} \frac{|D|}{|\{d' \in D : w_i \in d'\}|}$$

```
1 aargh<fff>3.23382650162
2 ahmet<fff>3.23382650162
3 xvt<fff>3.23382650162
4 unbvm1<fff>3.23382650162
5 deskwrit<fff>3.23382650162
6 oversight<fff>3.23382650162
7 coliseum<fff>3.23382650162
8 amorc<fff>3.23382650162
9 spacelab<fff>3.23382650162
10 brotherhood<fff>3.23382650162
11 blond<fff>3.23382650162
12 laden<fff>3.23382650162
13 dickinson<fff>3.23382650162
14 comedici<fff>3.23382650162
15 mje<fff>3.23382650162
16 durban<fff>3.23382650162
```

### ❑ Tiền xử lý: tính tf-idf

```
13 def get_tf_idf(data_path):
14     # get pre-computed idf values
15     with open('../datasets/20news-bydate/words_idfs.txt') as f:
16         words_idfs = [(line.split('<fff>')[0], float(line.split('<fff>')[1]))
17                         for line in f.read().splitlines()]
18
19     word_IDs = dict([(word, index)
20                       for index, (word, idf) in enumerate(words_idfs)])
21     idfs = dict(words_idfs)
22
23     with open(data_path) as f:
24         documents = [
25             (int(line.split('<fff>')[0]),
26              int(line.split('<fff>')[1]),
27              line.split('<fff>')[2])
28             for line in f.read().splitlines()]
```

```

30     data_tf_idf = []
31     for document in documents:
32         label, doc_id, text = document
33         words = [word for word in text.split() if word in idfs]
34         word_set = list(set(words))
35         max_term_freq = max([words.count(word)
36                               for word in word_set])
37

```

```

38     words_tfidfs = []
39     sum_squares = 0.0
40     for word in word_set:
41         term_freq = words.count(word)
42         tf_idf_value = term_freq * 1. / max_term_freq * idfs[word]
43         words_tfidfs.append((word_IDs[word], tf_idf_value))
44         sum_squares += tf_idf_value ** 2
45
46     words_tfidfs_normalized = [str(index) + ':'
47                                + str(tf_idf_value / np.sqrt(sum_squares))
48                                for index, tf_idf_value in words_tfidfs]
49
50     sparse_rep = ' '.join(words_tfidfs_normalized)
51     data_tf_idf.append((label, doc_id, sparse_rep))

```

### □ Tiền xử lý: Ghi data\_tf\_idf ra file

```

1 0<fff>49960<fff>9370:0.0967868358288 8553:0.229793603446 7347:0.132
2 0<fff>51060<fff>6637:0.0264613146274 9930:0.0291134757707 2501:0.03
3 0<fff>51119<fff>8648:0.16054261467 5695:0.211258519544 9671:0.62138
4 0<fff>51120<fff>10288:0.127635484843 9671:0.372830827488 10305:0.04
5 0<fff>51121<fff>7763:0.538997270778 10255:0.195617964114 9473:0.400
6 0<fff>51122<fff>10030:0.0719050870906 10174:0.0610246365229 3930:0.1
7 0<fff>51123<fff>10164:0.7012549744 10288:0.170180646457 9410:0.5434
8 0<fff>51124<fff>8717:0.172847984547 5979:0.225946768213 8475:0.7192
9 0<fff>51125<fff>9370:0.137114684091 9004:0.150470242337 10022:0.209
10 0<fff>51126<fff>10243:0.205727965198 9511:0.395948539607 8294:0.506
11 0<fff>51127<fff>9385:0.546772136773 10235:0.285813329788 10261:0.74
12 0<fff>51128<fff>9868:0.341744976285 10054:0.297276273338 9333:0.417
13 0<fff>51130<fff>10160:0.211947851892 9241:0.341062262272 10255:0.31
14 0<fff>51131<fff>10288:0.145869125535 10058:0.169030324161 9977:0.18
15 0<fff>51132<fff>9370:0.274229368182 10288:0.0850903232285 10135:0.1

```

## 2.2. Word2Vec

### 2.2.1. Reference

#### Machine Learning cho dữ liệu dạng bảng

Embedding là một kỹ thuật đưa một vector có số chiều lớn, thường ở dạng thưa, về một vector có số chiều nhỏ, thường ở dạng dày đặc.

Phương pháp này đặc biệt hữu ích với những đặc trưng hạng mục có số

🔗 [https://machinelearningcoban.com/tabcml\\_book/ch\\_embedding/embedding.html](https://machinelearningcoban.com/tabcml_book/ch_embedding/embedding.html)

	Không gian one-hot	Không gian embedding
Hà Nội	1 0 0 0 0 0	0.8 0.8
Hải Phòng	0 1 0 0 0 0	0.2 0.7
Tp HCM	0 0 1 0 0 0	0.9 -0.8
Bình Dương	0 0 0 1 0 0	0.25 -0.7
Hà Giang	0 0 0 0 1 0	0.08 1
Sóc Trăng	0 0 0 0 0 1	0.12 -0.9

### Machine Learning cho dữ liệu dạng bảng

Word2vec là một mô hình đơn giản và nổi tiếng giúp tạo ra các biểu diễn embedding của từ trong một không gian có số chiều thấp hơn nhiều lần so với số từ trong từ điển. Ý tưởng của word2vec đã được sử

❖ [https://machinelearningcoban.com/tablml\\_book/ch\\_embedding/wor d2vec.html](https://machinelearningcoban.com/tablml_book/ch_embedding/wor d2vec.html)

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

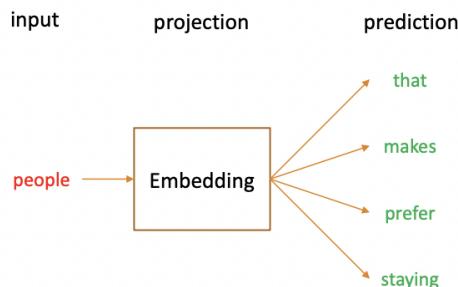
<https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf>

💡 <https://arxiv.org/pdf/1411.2738.pdf>

## 2.2.2. Word2Vec Model

### 1. Skip-Gram

❑ Sử dụng **center word** làm **input** và **context words** làm **target**



### 2. CBOW

❑ Sử dụng **context words** làm **input** và **center word** làm **target**

