

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

————— * —————

PROJECT 2

**Tìm hiểu một số thuật toán và mô hình học máy, học
sâu**

Tên: Lê Thành Chỉnh

Lớp: 727524

Giáo viên hướng dẫn: Trịnh Văn Loan

Mục lục

Chương 1: Linear Regression.....	4
1.Linear Regression là gì?	4
2. Bài toán hồi quy:.....	5
3. Triển khai thuật toán:	7
3.1 Dữ liệu sử dụng:.....	7
3.2 Cross-validation:.....	7
3.3 Triển khai:.....	8
3.4 Tiền xử lý dữ liệu:	13
Chương 2: Kmeans	21
1. Kmeans là gì?	21
2. Thuật toán:	22
3. Triển khai:	23
Chương 3: Multi-layer Perceptron	31
1. Tensorflow	31
2. Multi-layer Perceptron	34
2.1 Multi-layer Perceptron là gì?	34
2.2 Triển khai:.....	35
Tài liệu tham khảo	44

Lời mở đầu

Trong thời đại của cuộc cách mạng công nghệ 4.0, lĩnh vực học máy và học sâu đã nhanh chóng trở thành những nguồn cảm hứng sáng tạo và phát triển không ngừng cho cộng đồng khoa học và công nghệ. Những khả năng vượt bậc của các thuật toán và mô hình trong lĩnh vực này đã góp phần tạo ra những ứng dụng đột phá, từ phân tích dữ liệu đến trí tuệ nhân tạo, từ y học đến ô tô tự hành.

Đồ án này được thực hiện nhằm mục đích tìm hiểu và nắm vững một số thuật toán và mô hình quan trọng trong học máy và học sâu. Việc nắm bắt sâu sắc về cách hoạt động, ưu điểm và hạn chế của các thuật toán và mô hình này sẽ là chìa khóa để áp dụng chúng một cách hiệu quả vào các bài toán thực tế.

Qua quá trình nghiên cứu và thực hành, chúng tôi đã đạt được cái nhìn sâu hơn về sự phức tạp và sự thú vị của học máy và học sâu. Sự kết hợp giữa kiến thức lý thuyết và kỹ năng thực tế trong việc xây dựng và đánh giá các mô hình đã mang lại cho chúng tôi nhiều bài học quý báu.

Chúng tôi xin chân thành cảm ơn sự hướng dẫn và hỗ trợ của giáo viên hướng dẫn cũng như sự động viên từ bạn bè và gia đình trong suốt thời gian thực hiện đồ án này. Hy vọng rằng bài viết sau đây sẽ truyền tải được phần nào kiến thức và cảm xúc mà chúng tôi đã trải qua trong hành trình tìm hiểu về học máy và học sâu.

Trân trọng

Chương 1: Linear Regression

1. Linear Regression là gì?

Linear Regression là một phương pháp trong thống kê và máy học để mô hình hóa mối quan hệ tuyến tính giữa các biến. Đây là một phương pháp cơ bản và quan trọng trong việc dự đoán và phân tích dữ liệu. Dưới đây là một số điểm cơ bản về lý thuyết Linear Regression:

Khái niệm: Linear Regression (Hồi quy tuyến tính) là một phương pháp thống kê để tìm một mô hình tuyến tính giữa biến độc lập (X) và biến phụ thuộc (Y) sao cho mối quan hệ này có thể được sử dụng để dự đoán giá trị của Y dựa trên giá trị của X.

Công thức: Mô hình Linear Regression thường được biểu diễn bằng công thức: $Y = \beta_0 + \beta_1 X + \varepsilon$, trong đó Y là biến phụ thuộc, X là biến độc lập, β_0 là hệ số chặn của đường thẳng hồi quy, β_1 là hệ số góc của đường thẳng hồi quy, và ε biểu thị sai số ngẫu nhiên.

Mục tiêu: Mục tiêu của Linear Regression là tìm các hệ số β_0 và β_1 sao cho tổng bình phương của sai số ε là nhỏ nhất (phương pháp bình phương tối thiểu).

Phân tích sai số: Có nhiều cách để đánh giá mô hình Linear Regression, bao gồm SSE (Sum of Squared Errors), R-squared (hệ số xác định), và nhiều chỉ số khác.

Tiến hóa: Linear Regression có thể mở rộng để xử lý các biến đa chiều (Multiple Linear Regression), và cũng có thể áp dụng các biến đổi để tạo ra các phiên bản phức tạp hơn như Polynomial Regression.

Giả định: Một số giả định của Linear Regression bao gồm: sự tương quan tuyến tính giữa các biến, độc lập tuyệt đối giữa các sai số (điều kiện không tương quan), phân phối chuẩn của sai số, và độc lập của sai số.

Áp dụng: Linear Regression được sử dụng rộng rãi trong nhiều lĩnh vực, bao gồm kinh tế học, thống kê, khoa học xã hội, khoa học dữ liệu và machine learning.

2. Bài toán hồi quy:

Cho tập dữ liệu $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, trong đó mỗi điểm dữ liệu (x_i, y_i) bao gồm 2 thành phần:

$x_i = [x_{i1}, x_{i2}, \dots, x_{iK}]^T$ là một vector K chiều

$y_i \in \mathbb{R}$ là một số thực

Giả thiết rằng tồn tại hàm f tuyến tính sao cho $y_i \cong f(x_i)$:

$$f(x_i) = w_0 + w_1 x_{i1} + \dots + w_K x_{iK} = w x_i$$

Lỗi trên tập dữ liệu D :

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

Nghiệm w^* tối thiểu hóa L :

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

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:

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

Matrix algebra

Operation	Input	Complexity
Matrix multiplication	Two $n \times n$ matrices	$O(n^{2.373})$
Matrix multiplication	One $n \times m$ matrix & one $m \times p$ matrix	$O(nmp)$
Matrix inversion	One $n \times n$ 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

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:

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

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.

Hiệu 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

Có thể sử dụng các phương pháp tối ưu dựa trên gradient để minimize hàm lỗi. Áp dụng khi kích thước dữ liệu quá lớn việc tính nghịch đảo ma trận quá tốn kém

Lược đồ tối ưu:

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

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

3. Triển khai thuật toán:

3.1 Dữ liệu sử dụng:

Có tất cả 60 điểm dữ liệu, mỗi điểm dữ liệu có 15 thuộc tính và 1 giá trị death rate tương ứng

index	A1	A2	...	A13	A14	A15	Death Rate
1	36	27	...	15	59	59	921.870
2	35	23	...	10	39	57	997.875
3	44	29	...	6	33	54	962.354
4	47	45	...	8	24	56	982.291
...	

3.2 Cross-validation:

Một tập dữ liệu D thường có 2 phần: D_{train} và D_{test}

D_{train} dùng để huấn luyện mô hình

D_{test} để đánh giá hiệu quả của mô hình

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

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 ($i = 1, 2, 3, 4, 5$), ta thực hiện:

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

3.3 Triển khai:

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

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

Chuẩn hóa dữ liệu:

- > Các features có miền giá trị lệch nhau
- > Chuẩn hóa để đưa về 1 miền chung
- > Có nhiều phương pháp, 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

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

=> Ta cần thêm feature $x_{i0} = 1$ vào mỗi điểm dữ liệu

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

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

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

```

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	1030.380
7	43	30	74	• • •	32	62	56	934.700
8	45	30	73	• • •	4	4	56	899.529

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

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):...

```

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

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

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

> Hàm fit

```

44 def fit(self, X_train, Y_train, LAMBDA):
45     assert len(X_train.shape) == 2 and \
46            X_train.shape[0] == Y_train.shape[0]
47
48     W = np.linalg.inv(
49         X_train.transpose().dot(X_train) +
50         LAMBDA * np.identity(X_train.shape[1])
51     ).dot(X_train.transpose()).dot(Y_train)
52     return W

```

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

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

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

> 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

```

> 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
```

$$Y_{\text{new}} = X_{\text{new}}W$$
$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1K} \\ 1 & x_{21} & x_{22} & \dots & x_{2K} \\ \dots & & & & \\ 1 & x_{N1} & x_{N2} & \dots & x_{NK} \end{pmatrix}$$
$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}, W = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_K \end{pmatrix}$$

> Hàm compute_RSS

```
59 def compute_RSS(self, Y_new, Y_predicted):  
60     loss = 1. / Y_new.shape[0] * \  
61         np.sum((Y_new - Y_predicted) ** 2)  
62     return loss
```

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

> 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

> Hàm get_the_best_LAMBDA

```
64 def get_the_best_LAMBDA(self, X_train, Y_train):
65     def cross_validation(num_folds, LAMBDA):...
78
79     def range_scan(best_LAMBDA, minimum_RSS, LAMBDA_values):...
86
87     best_LAMBDA, minimum_RSS = range_scan(best_LAMBDA=0, minimum_RSS=10000 ** 2,
88                                           LAMBDA_values=range(50)) # [0, 1, 2, ..., 49]
89
90     LAMBDA_values = [k * 1. / 1000 for k in range(
91                     max(0, (best_LAMBDA - 1) * 1000), (best_LAMBDA + 1) * 1000, 1)
92                     ] # step size = 0.001
93
94     best_LAMBDA, minimum_RSS = range_scan(best_LAMBDA=best_LAMBDA, minimum_RSS=minimum_RSS,
95                                           LAMBDA_values=LAMBDA_values)
96     return best_LAMBDA
```

>Hàm range_scan

```
79 def range_scan(best_LAMBDA, minimum_RSS, LAMBDA_values):
80     for current_LAMBDA in LAMBDA_values:
81         aver_RSS = cross_validation(num_folds=5, LAMBDA=current_LAMBDA)
82         if aver_RSS < minimum_RSS:
83             best_LAMBDA = current_LAMBDA
84             minimum_RSS = aver_RSS
85     return best_LAMBDA, minimum_RSS
```

>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
```

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)
```

3.4 Tiền xử lý dữ liệu:

3.4.1 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 $\text{tf-idf}(w_i, d, D)$ được tính như sau:

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

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

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

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

Biểu diễn TF-IDF

Xác định từ điển V:

Với mỗi văn bản d trong D:

* B1: Tách d thành các từ theo punctuations ta thu được W_d :

‘Data-Science Lab;2018’ -> [‘Data’, ‘Science’, ‘Lab’, ‘2018’]

* B2: Loại bỏ từ dừng (stop words) khỏi W_d :

$$W_d = W_d \setminus \{\text{stop_words}\}$$

a, an, the, have, for,

* B3: Đưa các từ về dạng gốc (stemming) :

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

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

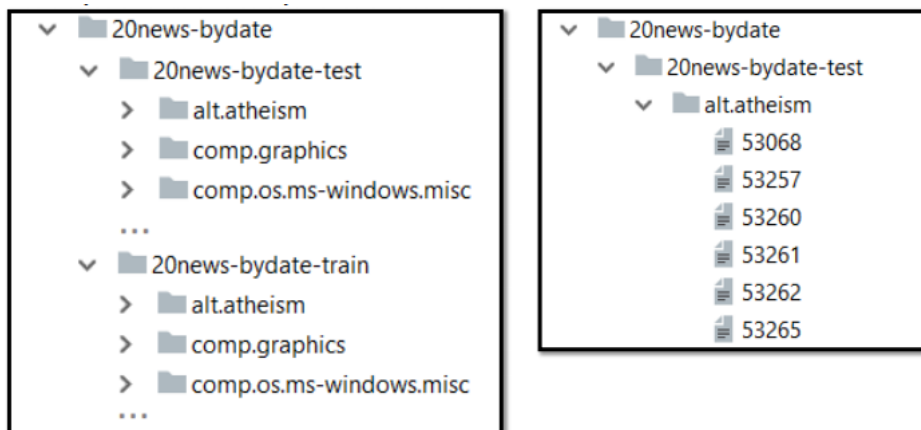
Cuối cùng, ta có:

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

Tập dữ liệu sử dụng: 20newsgroups

Bao gồm xấp xỉ 20,000 bài báo, thuộc 20 nhóm tin tức khác nhau.

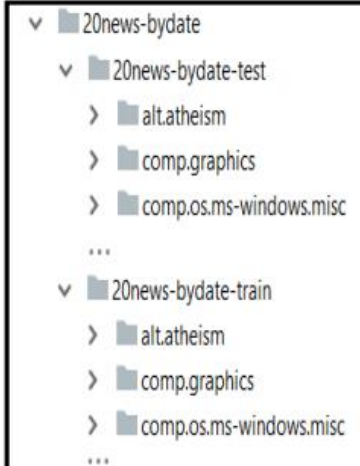
cấu trúc cây thư mục:



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**


```

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

```


> 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
```

```
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]))
```

```
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 comedi<fff>3.23382650162
15 mje<fff>3.23382650162
16 durban<fff>3.23382650162
```

>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_tfidf = []
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_tfidf.append((word_IDs[word], tf_idf_value))
44     sum_squares += tf_idf_value ** 2
45
46 words_tfidf_normalized = [str(index) + ':'
47                            + str(tf_idf_value / np.sqrt(sum_squares))
48                            for index, tf_idf_value in words_tfidf]
49
50 sparse_rep = ' '.join(words_tfidf_normalized)
51 data_tf_idf.append((label, doc_id, sparse_rep))
```

>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.
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 10022:0.205
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
```

3.4.2 Word2vec: Biểu diễn vector cho từ:

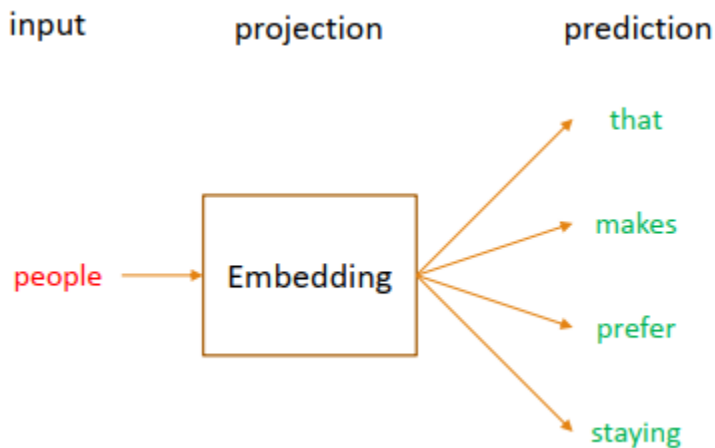
Để thu được biểu diễn word2vec của từ, có 2 mô hình:

1. Skip-Gram

2. CBOW (Continuous Bag-of-Word Model)

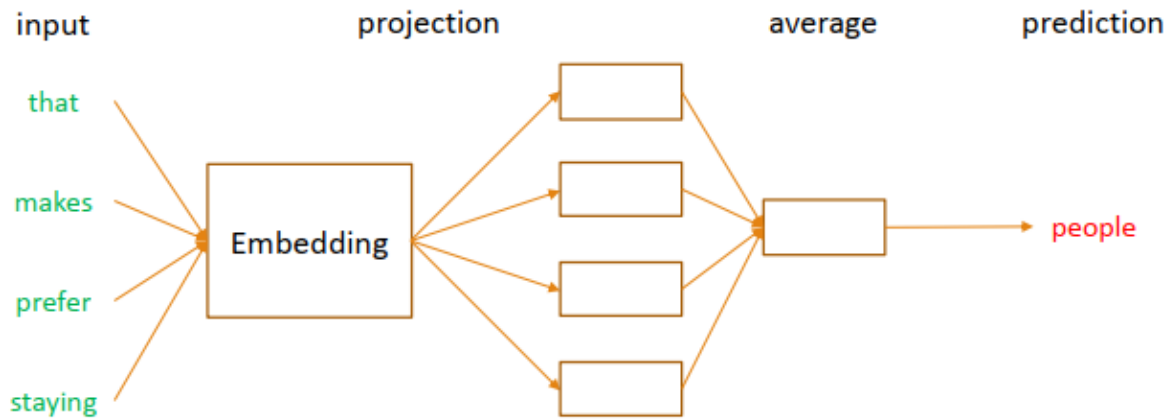
3.4.2.1. Skip-Gram:

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



3.4.2.2 CBOW

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



Sau khi huấn luyện, thu được ma trận **Word Embedding**, mỗi hàng là một vector biểu diễn cho một từ.

Lưu ý: ma trận Word Embedding cũng thay đổi khi training

Word Embedding thường là tầng đầu tiên trong rất nhiều mô hình Deeplearning hiện nay

Chương 2: Kmeans

1. Kmeans là gì?

K-means là một thuật toán trong lĩnh vực máy học và thống kê được sử dụng để phân cụm dữ liệu. Thuật toán này giúp tách một tập dữ liệu thành các nhóm (clusters) dựa trên các đặc điểm tương tự của các điểm dữ liệu. Mục tiêu của K-means là tìm ra các điểm trung tâm của các nhóm sao cho tổng bình phương khoảng cách từ các điểm dữ liệu tới điểm trung tâm gần nhất là nhỏ nhất.

Cách hoạt động của K-means như sau:

Khởi tạo các điểm trung tâm ban đầu của các nhóm (thường là ngẫu nhiên hoặc dựa trên kiến thức tiên định).

Gán từng điểm dữ liệu vào nhóm có điểm trung tâm gần nhất.

Cập nhật lại điểm trung tâm của mỗi nhóm dựa trên các điểm dữ liệu đã được gán vào nhóm đó.

Lặp lại bước 2 và 3 cho đến khi không có sự thay đổi đáng kể trong việc gán điểm dữ liệu và cập nhật điểm trung tâm.

Kết quả cuối cùng của thuật toán là các nhóm dữ liệu có đặc điểm tương tự được phân chia dựa trên sự tối ưu hóa khoảng cách giữa điểm dữ liệu và điểm trung tâm của nhóm. K-means thường được sử dụng trong nhiều ứng dụng khác nhau như phân cụm dữ liệu, nén ảnh, phân loại và khám phá dữ liệu.

2. Thuật toán:

Input:

- > Tập dữ liệu $R = \{r_d : d \in D\}$ với $r_d \in \mathbb{R}^{|V|}$ là biểu diễn tf-idf của d
- > Số cụm K

Output: $A = \{a_d : d \in D\}$ với $a_d \in \{1, 2, \dots, K\}$ cho biết d được phân vào cụm nào

Procedure:

- > **B1:** Khởi tạo tâm cho K cụm:

$E = \{e_k\}$ với e_k là tâm của cụm k ,

$k \in \{1, 2, \dots, K\}$ và $|E| = K$,

E là 1 tập con gồm K phần tử được lấy từ $R = \{r_d : d \in D\}$

- > **B2:** Lặp cho tới khi hội tụ:

- * Với mỗi $d \in D$:

- + Tính $\text{similarity}(r_d, e_k)$

- + Gán d vào cụm k^* với $k^* = \text{argmax}_k(\text{similarity}(r_d, e_k))$

- * Cập nhật lại E

Lựa chọn điều kiện dừng:

- > Số bước lặp vượt quá 1 ngưỡng đặt trước: $\text{iteration} > \text{max_iters}$

- > $E = \{e_k\}$ thay đổi không đáng kể:

$$|E_{\text{new}} \setminus E_{\text{old}}| < n_0 \text{ với } n_0 \ll K$$

- > Độ tương đồng trung bình không tăng hoặc tăng không đáng kể

- * Độ giảm lỗi phân cụm: $S_{\text{new}} - S_{\text{old}} < \varepsilon$

- * Lỗi phân cụm: $S = \frac{1}{|D|} \sum_{r_d \in D} \text{similarity}(r_d, e_{a_d})$

Đánh giá chất lượng phân cụm:

> **Purity:**

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{k=1, K} \max_j |\omega_k \cap c_j|$$

với $\omega_k = \{d: a_d = k, d \in D\}$

và $c_j = \{d: \text{label}(d) = j, d \in D\}$

$\omega_k \cap c_j$: tập hợp các văn bản trong cụm k có nhãn j

> **NMI (normalized mutual information):**

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega, \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2} \text{ với } \mathbb{C} = \{c_0, c_1, \dots, c_{J-1}\}, J \text{ là số lớp}$$

$$\text{với } I(\Omega, \mathbb{C}) = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \cdot \log_{10} \frac{N \cdot |\omega_k \cap c_j|}{|\omega_k| \cdot |c_j|}$$

$$H(\Omega) = - \sum_k \frac{|\omega_k|}{N} \cdot \log_{10} \frac{|\omega_k|}{N}$$

$$H(\mathbb{C}) = - \sum_k \frac{|c_j|}{N} \cdot \log_{10} \frac{|c_j|}{N}$$

3. Triển khai:

20newsgroups dataset đã tiền xử lý:

```
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.
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 10022:0.205
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 10255:0.33
14 0<fff>51131<fff>10288:0.145869125535 10058:0.169030324161 9977:0.16
15 0<fff>51132<fff>9370:0.274229368182 10288:0.0850903232285 10135:0.1
```

Mỗi cụm ta lưu trữ các thông tin sau:

- > centroid: tâm cụm
- > members: danh sách các điểm dữ liệu trong cụm

Mỗi điểm dữ liệu d ta sẽ lưu trữ thông tin sau:

- > r_d : biểu diễn tf-idf r_d của văn bản d
- > label: newsgroup của văn bản d
- > doc_id: tên file chứa văn bản d

Ta sẽ xây dựng 3 lớp:

- > 2 lớp cho lưu trữ thông tin:
 - * class Cluster
 - * class Member
- > 1 lớp Kmeans cho triển khai thuật toán
 - * class Kmeans

class Member:

```
160 class Member:
161     def __init__(self, r_d, label=None, doc_id=None):
162         self._r_d = r_d
163         self._label = label
164         self._doc_id = doc_id
```

class Cluster:

```
148 class Cluster:
149     def __init__(self):
150         self._centroid = None
151         self._members = []
152
153     def reset_members(self):
154         self._members = []
155
156     def add_member(self, member):
157         self._members.append(member)
```


class Kmeans:

```
167 class Kmeans:
168     def __init__(self, num_clusters):...
174
175     def load_data(self, data_path):...
203
204     def random_init(self, seed_value):...
215     def compute_similarity(self, member, centroid):...
217     def select_cluster_for(self, member):...
228     def update_centroid_of(self, cluster):...
235     def stopping_condition(self, criterion, threshold):...
259     def run(self, seed_value, criterion, threshold):...
288
289     def compute_purity(self):...
296     def compute_NMI(self):...
```

Hàm khởi tạo:

```
167 class Kmeans:
168     def __init__(self, num_clusters):
169         self._num_clusters = num_clusters
170         self._clusters = [Cluster() for _ in
171                           range(self._num_clusters)]
172         self._E = [] # list of centroids
173         self._S = 0 # overall similarity
```

Đọc dữ liệu:

```
175     def load_data(self, data_path):
176         def sparse_to_dense(sparse_r_d, vocab_size):...
183
184         with open(data_path) as f:
185             d_lines = f.read().splitlines()
186         with open('../datasets/20news-bydate/words_idfs.txt') as f:
187             vocab_size = len(f.read().splitlines())
188
189         self._data = []
190         self._label_count = defaultdict(int)
191         for data_id, d in enumerate(d_lines):
192             features = d.split('<fff>')
193             label, doc_id = int(features[0]), int(features[1])
194             self._label_count[label] += 1
195             r_d = sparse_to_dense(sparse_r_d=features[2], vocab_size=vocab_size)
196
197             self._data.append(Member(r_d=r_d, label=label, doc_id=doc_id))
```

Đọc dữ liệu: Hàm `sparse_to_dense`

```
176 def sparse_to_dense(sparse_r_d, vocab_size):
177     r_d = [0.0 for _ in range(vocab_size)]
178     indices_tfidf = sparse_r_d.split()
179     for index_tfidf in indices_tfidf:
180         index, = int(index_tfidf.split(':')[0])
181         tfidf = float(index_tfidf.split(':')[1])
182         r_d[index] = tfidf
183     return np.array(r_d)
```

Chạy thuật toán: Hàm `run`

```
262 def run(self, seed_value, criterion, threshold):
263     self.random_init(seed_value)
264
265     # continually update clusters until convergence
266     self._iteration = 0
267     while True:
268         # reset clusters, retain only centroids
269         for cluster in self._clusters:
270             cluster.reset_members()
271         self._new_S = 0
272         for member in self._data:
273             max_s = self.select_cluster_for(member)
274             self._new_S += max_s
275         for cluster in self._clusters:
276             self.update_centroid_of(cluster)
277
278         self._iteration += 1
279         if self.stopping_condition(criterion, threshold):
280             break
```

Chạy thuật toán: xác định cụm cho từng điểm dữ liệu

```
217 def select_cluster_for(self, member):
218     best_fit_cluster = None
219     max_similarity = -1
220     for cluster in self._clusters:
221         similarity = self.compute_similarity(member, cluster._centroid)
222         if similarity > max_similarity:
223             best_fit_cluster = cluster
224             max_similarity = similarity
225
226     best_fit_cluster.add_member(member)
227     return max_similarity
```

Chạy thuật toán: cập nhật lại tâm cụm

```
229 def update_centroid_of(self, cluster):
230     member_r_ds = [member._r_d for member in cluster._members]
231     aver_r_d = np.mean(member_r_ds, axis=0)
232     sqrt_sum_sqr = np.sqrt(np.sum(aver_r_d ** 2))
233     new_centroid = np.array([value / sqrt_sum_sqr for value in aver_r_d])
234
235     cluster._centroid = new_centroid
```

Chạy thuật toán: Kiểm tra điều kiện dừng – max_iters

```
237 def stopping_condition(self, criterion, threshold):
238     criteria = ['centroid', 'similarity', 'max_iters']
239     assert criterion in criteria
240     if criterion == 'max_iters':
241         if self._iteration >= threshold:
242             return True
243         else:
244             return False
```

Chạy thuật toán: Kiểm tra điều kiện dừng – centroid

```
245 elif criterion == 'centroid':
246     E_new = [list(cluster._centroid) for cluster in self._clusters]
247     E_new_minus_E = [centroid for centroid in E_new
248                     if centroid not in self._E]
249     self._E = E_new
250     if len(E_new_minus_E) <= threshold:
251         return True
252     else:
253         return False
```

Chạy thuật toán: Kiểm tra điều kiện dừng – similarity

```
254 else:
255     new_S_minus_S = self._new_S - self._S
256     self._S = self._new_S
257     if new_S_minus_S <= threshold:
258         return True
259     else:
260         return False
```

```
271 self._new_S = 0
272 for member in self._data:
273     max_s = self.select_cluster_for(member)
274     self._new_S += max_s
```

Đánh giá chất lượng phân cụm: Tính purity

```
282 def compute_purity(self):
283     majority_sum = 0
284     for cluster in self._clusters:
285         member_labels = [member._label for member in cluster._members]
286         max_count = max([member_labels.count(label) for label in range(20)])
287         majority_sum += max_count
288     return majority_sum * 1. / len(self._data)
```

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{k=1, K} \max_j |\omega_k \cap c_j|$$

Đánh giá chất lượng phân cụm: Tính NMI

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega, \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2}$$

$$I(\Omega, \mathbb{C}) = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \cdot \log_{10} \frac{N \cdot |\omega_k \cap c_j|}{|\omega_k| \cdot |c_j|}$$

$$H(\Omega) = - \sum_k \frac{|\omega_k|}{N} \cdot \log_{10} \frac{|\omega_k|}{N}$$

$$H(\mathbb{C}) = - \sum_k \frac{|c_j|}{N} \cdot \log_{10} \frac{|c_j|}{N}$$

```
290 def compute_NMI(self):
291     I_value, H_omega, H_C, N = 0., 0., 0., len(self._data)
292     for cluster in self._clusters:
293         wk = len(cluster._members) * 1.
294         H_omega += - wk / N * np.log10(wk / N)
295         member_labels = [member._label
296                           for member in cluster._members]
297         for label in range(20):
298             wk_cj = member_labels.count(label) * 1.
299             cj = self._label_count[label]
300             I_value += wk_cj / N * \
301                 np.log10(N * wk_cj / (wk * cj) + 1e-12)
302     for label in range(20):
303         cj = self._label_count[label] * 1.
304         H_C += - cj / N * np.log10(cj / N)
305     return I_value * 2. / (H_omega + H_C)
```

Vấn đề khởi tạo tâm cụm:

> Kết quả của Kmeans phụ thuộc vào việc khởi tạo tâm cụm

=> Làm vài lần và chọn lấy lần tốt nhất

hoặc Khởi tạo theo chiến lược:

*** Dùng Kmeans++**

*** Cluster center initialization algorithm for Kmeans clustering**

Sử dụng Scikit-learn

>Kmeans

>SVMs: > Linear SVMs

> kernel SVMs

Kmeans:

```
63 def clustering_with_KMeans():
64     data, labels = load_data(data_path='../datasets/20news-bydate/20news-full-tfidf.txt')
65     # use csr_matrix to create a sparse matrix with efficient row slicing
66     from sklearn.cluster import KMeans
67     from scipy.sparse import csr_matrix
68     X = csr_matrix(data)
69     print '======'
70     kmeans = KMeans(
71         n_clusters=20,
72         init='random',
73         n_init=5, # number of time that kmeans runs with differently initialized centroids
74         tol=1e-3, # threshold for acceptable minimum error decrease
75         random_state=2018 # set to get deterministic results
76     ).fit(X)
77     labels = kmeans.labels_
```

SVMs: Linear SVMs:

```
34 def classifying_with_linear_SVMs():
35     train_X, train_y = load_data(data_path='../datasets/20news-bydate/20news-train-tfidf.txt')
36     from sklearn.svm import LinearSVC
37     classifier = LinearSVC(
38         C=10.0, # penalty coeff
39         tol=0.001, # tolerance for stopping criteria
40         verbose=True # whether prints out logs or not
41     )
42     classifier.fit(train_X, train_y)
43
44     test_X, test_y = load_data(data_path='../datasets/20news-bydate/20news-test-tfidf.txt')
45     predicted_y = classifier.predict(test_X)
46     accuracy = compute_accuracy(predicted_y=predicted_y, expected_y=test_y)
47     print 'Accuracy:', accuracy
```

Hàm compute_accuracy

```
28 def compute_accuracy(predicted_y, expected_y):  
29     matches = np.equal(predicted_y, expected_y)  
30     accuracy = np.sum(matches.astype(float)) / expected_y.size  
31     return accuracy
```

Kernel SVMs:

```
60 classifier = SVC(  
61     C=50.0,  
62     kernel='rbf', # 'linear', 'poly', 'rbf', 'sigmoid', precomputed'  
63     gamma=0.1,  
64     tol=0.001,  
65     verbose=True  
66 )
```

radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$

Chương 3: Multi-layer Perceptron

1. Tensorflow

Tensorflow là gì – Với sự bùng nổ của lĩnh vực Trí Tuệ Nhân Tạo – A.I. trong thập kỷ vừa qua, machine learning và deep learning rõ ràng cũng phát triển theo cùng. Và ở thời điểm hiện tại, TensorFlow chính là thư viện mã nguồn mở cho machine learning nổi tiếng nhất thế giới, được phát triển bởi các nhà nghiên cứu từ Google. Việc hỗ trợ mạnh mẽ các phép toán học để tính toán trong machine learning và deep learning đã giúp việc tiếp cận các bài toán trở nên đơn giản, nhanh chóng và tiện lợi hơn nhiều.

Các hàm được dựng sẵn trong thư viện cho từng bài toán cho phép TensorFlow xây dựng được nhiều neural network. Nó còn cho phép bạn tính toán song song trên nhiều máy tính khác nhau, thậm chí trên nhiều CPU, GPU trong cùng 1 máy hay tạo ra các dataflow graph – đồ thị luồng dữ liệu để dựng nên các model. Nếu bạn muốn chọn con đường sự nghiệp trong lĩnh vực A.I. này, nắm rõ những điều cơ bản của TensorFlow thực sự rất quan trọng.

Được viết bằng C++ và thao tác interface bằng Python nên phần performance của TensorFlow cực kỳ tốt. Đối tượng sử dụng nó cũng đa dạng không kém: từ các nhà nghiên cứu, nhà khoa học dữ liệu và dĩ nhiên không thể thiếu các lập trình viên.

Kiến trúc của TensorFlow:

Kiến trúc TensorFlow hoạt động được chia thành 3 phần:

- Tiền xử lý dữ liệu**
- Dựng model**
- Train và ước tính model**

Cách TensorFlow hoạt động

TensorFlow cho phép các lập trình viên tạo ra dataflow graph, cấu trúc mô tả làm thế nào dữ liệu có thể di chuyển qua 1 biểu đồ, hay 1 sê-ri các node đang xử lý. Mỗi node trong đồ thị đại diện 1 operation toán học, và mỗi kết nối hay edge giữa các node là 1 mảng dữ liệu đa chiều, hay còn được gọi là ‘tensor’.

TensorFlow cung cấp tất cả những điều này cho lập trình viên theo phương thức của ngôn ngữ Python. Vì Python khá dễ học và làm việc, ngoài ra còn cung cấp nhiều cách tiện lợi để ta hiểu được làm thế nào các high-level abstractions có thể kết hợp cùng nhau. Node và tensor trong TensorFlow là các đối tượng Python, và các ứng dụng TensorFlow bản thân chúng cũng là các ứng dụng Python.

Các operation toán học thực sự thì không được thi hành bằng Python. Các thư viện biến đổi có sẵn thông qua TensorFlow được viết bằng các binary C++ hiệu suất cao. Python chỉ điều hướng lưu lượng giữa các phần và cung cấp các high-level abstraction lập trình để nối chúng lại với nhau.

TensorFlow 2.0, được ra mắt vào tháng 10 năm 2019, cải tiến framework theo nhiều cách dựa trên phản hồi của người dùng, để dễ dàng và hiệu quả hơn khi làm việc cùng nó (ví dụ: bằng cách sử dụng các Keras API liên quan đơn giản cho việc train model). Train phân tán dễ chạy hơn nhờ vào API mới và sự hỗ trợ cho TensorFlow Lite cho phép triển khai các mô hình trên khá nhiều nền tảng khác nhau. Tuy nhiên, nếu đã viết code trên các phiên bản trước đó của TensorFlow thì bạn phải viết lại, đôi lúc 1 ít, đôi lúc cũng khá đáng kể, để tận dụng tối đa các tính năng mới của TensorFlow 2.0.

Lợi ích từ TensorFlow

Lợi ích dễ thấy nhưng quan trọng nhất mà TensorFlow cung cấp cho việc lập trình machine learning chính là abstraction. Thay vì phải đối phó với những tình huống rườm rà từ việc thực hiện triển khai các thuật toán, hay tìm ra cách hợp lý để chuyển output của 1 chức năng sang input của 1 chức năng khác, giờ đây bạn có thể tập trung vào phần logic tổng thể của 1 ứng dụng hơn. TensorFlow sẽ chăm sóc phần còn lại thay cho bạn.

Ngoài ra TensorFlow còn cung cấp các tiện ích bổ sung cho các lập trình viên cần debug cũng như giúp bạn tự suy xét các ứng dụng TensorFlow. Chế độ eager execution cho phép bạn đánh giá và sửa đổi từng operation của biểu đồ 1 cách riêng biệt và minh bạch, thay vì phải dựng toàn bộ biểu đồ dưới dạng 1 đối tượng độc lập vốn khá mơ hồ hay phải đánh giá chung tổng thể. Cuối cùng, 1 tính năng khá độc đáo của TensorFlow là TensorBoard. TensorBoard cho phép bạn quan sát 1 cách trực quan những gì TensorFlow đang làm.

TensorFlow còn có nhiều cải tiến từ sự hậu thuẫn từ các ekip thương mại hạng A tại Google. Google không những tiếp lửa cho tiến độ nhanh chóng cho sự phát triển đằng sau dự án, mà còn tạo ra nhiều phục vụ độc đáo xung quanh TensorFlow để nó dễ dàng deploy và sử dụng: như silicon TPU mình đã nói ở trên để tăng tốc hiệu suất đám mây Google, 1 online hub cho việc chia sẻ các model được tạo với framework, sự hiện diện của in-browser và gần gũi với mobile của framework, và nhiều hơn thế nữa...

Lưu ý: Trong 1 số công việc training, vài chi tiết về việc triển khai của TensorFlow làm cho nó khó có thể quyết định được hoàn toàn kết quả training model. Đôi khi 1 model được train trên 1 hệ thống này sẽ có thay đổi 1 chút so với 1 model được train trên hệ thống khác, ngay cả khi chúng được cung cấp dữ liệu như nhau. Các nguyên nhân cho điều này cũng xê xích hay 1 số hành vi khi không được xác định khi sử dụng GPU. Điều này nói rằng, các vấn đề đó có thể giải quyết được, và đội ngũ của TensorFlow cũng đang xem xét việc kiểm soát nhiều hơn để ảnh hưởng đến tính quyết định trong quy trình làm việc.

Các Component của TensorFlow

Tensor

Tên của TensorFlow được đưa ra trực tiếp là nhờ vào framework cốt lõi của nó: Tensor. Trong TensorFlow, tất cả các tính toán đều liên quan tới các tensor. 1 tensor là 1 vector hay ma trận của n-chiều không gian đại diện cho tất cả loại dữ liệu. Tất cả giá trị trong 1 tensor chứa đựng loại dữ liệu giống hệt nhau với 1 shape đã biết (hoặc đã biết 1 phần). Shape của dữ liệu chính là chiều của ma trận hay mảng.

1 tensor có thể được bắt nguồn từ dữ liệu input hay kết quả của 1 tính toán. Trong TensorFlow, tất cả các hoạt động được tiến hành bên trong 1 graph – biểu đồ. Biểu đồ là 1 tập hợp tính toán được diễn ra liên tiếp. Mỗi operation được gọi là 1 op node (operation node) và được kết nối với nhau.

Biểu đồ phát thảo các op và kết nối giữa các node. Tuy nhiên, nó không hiển thị các giá trị. Phần edge của các node chính là tensor, 1 cách để nhập operation với dữ liệu.

Graph

TensorFlow sử dụng framework dạng biểu đồ. Biểu đồ tập hợp và mô tả tất cả các chuỗi tính toán được thực hiện trong quá trình training. Biểu đồ cũng mang rất nhiều lợi thế:

- Nó được làm ra để chạy trên nhiều CPU hay GPU, ngay cả các hệ điều hành trên thiết bị điện thoại.
- Tính di động của biểu đồ cho phép bảo toàn các tính toán để bạn sử dụng ngay hay sau đó. Biểu đồ có thể được lưu lại để thực thi trong tương lai.
- Tất cả tính toán trong biểu đồ được thực hiện bằng cách kết nối các tensor lại với nhau. 1 tensor có 1 node và 1 edge. Node mang operation toán học và sản xuất các output ở đầu cuối. Các edge giải thích mối quan hệ input/output giữa các node.

2. Multi-layer Perceptron

2.1 Multi-layer Perceptron là gì?

Một Multi-Layer Perceptron (MLP) là một loại mạng neural feedforward, tức là thông tin di chuyển qua từ lớp đầu tiên đến lớp cuối cùng mà không có các vòng lặp hay kết nối ngược. MLP là một thành phần cơ bản của deep learning và được sử dụng rộng rãi trong nhiều ứng dụng khác nhau như xử lý ngôn ngữ tự nhiên, nhận dạng hình ảnh, và dự đoán dữ liệu.

MLP gồm có ít nhất ba lớp:

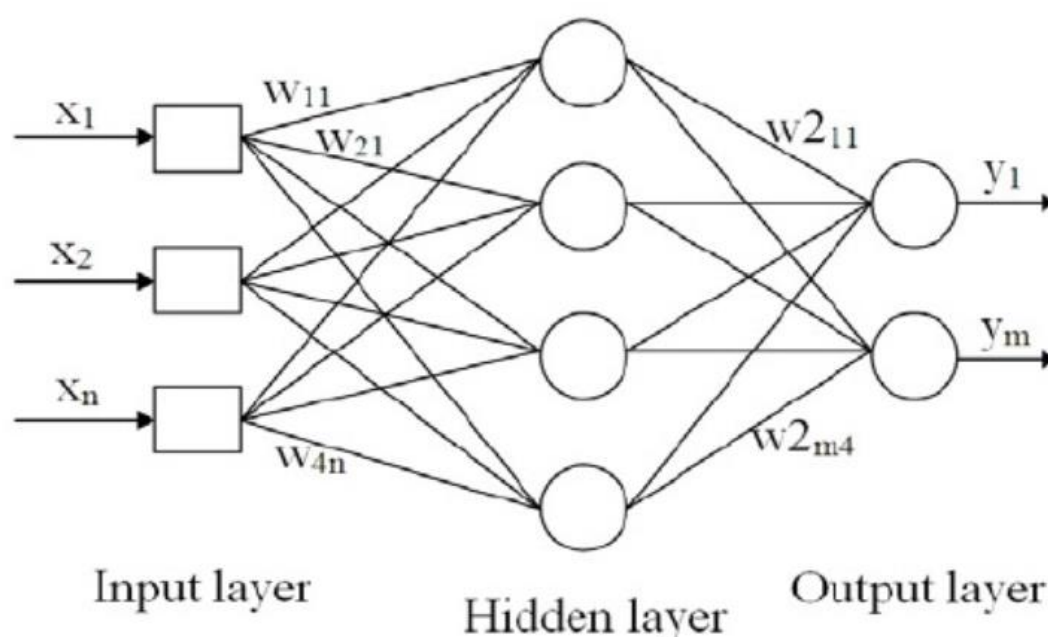
- Lớp đầu vào (Input Layer): Lớp này nhận các đặc trưng hoặc dữ liệu đầu vào của mô hình. Mỗi nút trong lớp này thể hiện một đặc trưng riêng biệt của dữ liệu.
- Các lớp ẩn (Hidden Layers): Đây là các lớp nằm giữa lớp đầu vào và lớp đầu ra. Mỗi lớp ẩn có một số nút (neurons) và mỗi nút kết nối với tất cả các nút trong lớp trước đó và sau đó. Các lớp ẩn này giúp mô hình học được các biểu diễn phức tạp từ dữ liệu.
- Lớp đầu ra (Output Layer): Lớp này cho ra kết quả của mô hình, ví dụ như dự đoán một lớp hoặc giá trị. Số nút trong lớp đầu ra thường phụ thuộc vào mục tiêu của bài toán (ví dụ, phân loại đa lớp sẽ có nhiều nút tương ứng với số lớp).

Các kết nối giữa các nút trong các lớp là trọng số (weights) và mô hình học chính là quá trình tối ưu hóa các trọng số này để dự đoán đầu ra chính xác nhất có thể.

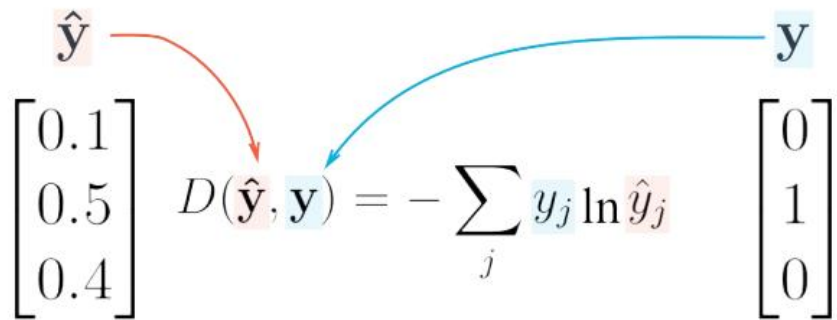
Để huấn luyện một MLP, bạn cần sử dụng các thuật toán tối ưu hóa như gradient descent để điều chỉnh trọng số dựa trên sai số giữa dự đoán và kết quả thực tế. Quá trình này được lặp lại qua nhiều lần cho đến khi mô hình hội tụ và có khả năng dự đoán tốt trên dữ liệu mới.

MLP có thể mở rộng để chứa nhiều lớp ẩn và nhiều nút, tạo thành các kiến trúc mạng neural sâu (deep neural networks), làm cho chúng có khả năng học biểu diễn phức tạp từ dữ liệu lớn hơn và giải quyết các bài toán khó khăn hơn.

2.2 Triển khai:



Hàm lỗi phân lớp:


$$\hat{\mathbf{y}} = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$
$$D(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_j y_j \ln \hat{y}_j$$

$$\Rightarrow \text{cross-entropy}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K y_k \cdot \log_k(\hat{y}_k)$$

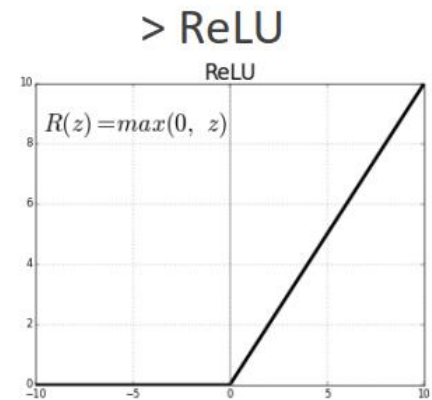
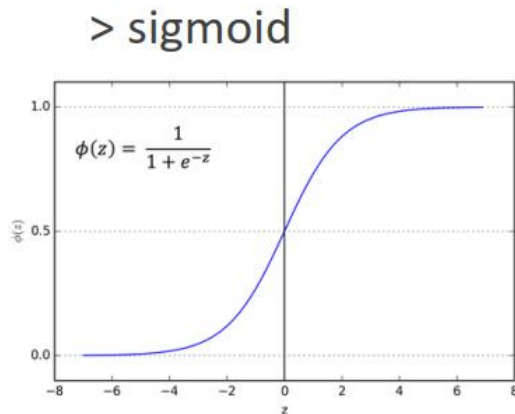
Mô hình hóa:

```
hidden = x.dot(W1)+b1    # X: [1, K] ; W1: [K, M] ; b1: [M,]
hidden = activation(hidden)    # [1, M]
logits = hidden.dot(W2)+b2    # [1, num_classes]
outputs = softmax(logits)    # [1, num_classes]
loss = cross-entropy(outputs, one-hot(real-label))
predicted-label =  $\underset{i}{\operatorname{argmax}}(\text{outputs}_i)$  # [1,]
```

Hàm softmax:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Hàm activation:



Các bước để xây dựng mô hình trên tensorflow:

- > B1: Xây dựng computation graph
- > B2: Mở một phiên làm việc (session), truyền (feed) dữ liệu vào graph và chạy

Xây dựng **class MLP**

```
69 class MLP:
70     def __init__(self, vocab_size, hidden_size):...
75
76     def build_graph(self):...
133
134     def trainer(self, loss, learning_rate):...
```

Xây dựng **class MLP**: hàm init

```
70     def __init__(self, vocab_size, hidden_size):
71         self._vocab_size = vocab_size
72         self._hidden_size = hidden_size
```

Xây dựng class **MLP**: hàm build_graph

```
74 def build_graph(self):
75     self._X = tf.placeholder(tf.float32, shape=[None, self._vocab_size])
76     self._real_Y = tf.placeholder(tf.int32, shape=[None, ])
77
78     weights_1 = tf.get_variable(
79         name='weights_input_hidden',
80         shape=(self._vocab_size, self._hidden_size),
81         initializer=tf.random_normal_initializer(seed=2018),
82     )
83     biases_1 = tf.get_variable(
84         name='biases_input_hidden',
85         shape=(self._hidden_size),
86         initializer=tf.random_normal_initializer(seed=2018)
87     )
```

Xây dựng class **MLP**: hàm build_graph

```
88     weights_2 = tf.get_variable(
89         name='weights_hidden_output',
90         shape=(self._hidden_size, NUM_CLASSES),
91         initializer=tf.random_normal_initializer(seed=2018),
92     )
93     biases_2 = tf.get_variable(
94         name='biases_hidden_output',
95         shape=(NUM_CLASSES),
96         initializer=tf.random_normal_initializer(seed=2018)
97     )
```

```
99     hidden = tf.matmul(self._X, weights_1) + biases_1
100     hidden = tf.sigmoid(hidden)
101     logits = tf.matmul(hidden, weights_2) + biases_2
102
103     labels_one_hot = tf.one_hot(indices=self._real_Y, depth=NUM_CLASSES,
104                                 dtype=tf.float32)
105     loss = tf.nn.softmax_cross_entropy_with_logits(labels=labels_one_hot,
106                                                    logits=logits)
107
108     loss = tf.reduce_mean(loss)
109     probs = tf.nn.softmax(logits)
110     predicted_labels = tf.argmax(probs, axis=1)
111     predicted_labels = tf.squeeze(predicted_labels)
112
113     return predicted_labels, loss
```

Xây dựng **class MLP**: hàm trainer: chọn thuật toán để tối ưu hàm Loss

```
115     def trainer(self, loss, learning_rate):
116         train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)
117         return train_op
```

Xây dựng computation graph

```
170         # create a computation graph
171         with open('../datasets/words_idfs.txt') as f:
172             vocab_size = len(f.read().splitlines())
173
174         mlp = MLP(
175             vocab_size=vocab_size,
176             hidden_size=50
177         )
178         predicted_labels, loss = mlp.build_graph()
179         train_op = mlp.trainer(loss=loss, learning_rate=0.1)
```

Mở một phiên làm việc, truyền dữ liệu và chạy

```
193         # open a session to run
194         with tf.Session() as sess:
195             train_data_reader, test_data_reader = load_dataset()
196             step, MAX_STEP = 0, 1000 ** 2
197
198             sess.run(tf.global_variables_initializer())
199             while step < MAX_STEP:
200                 train_data, train_labels = train_data_reader.next_batch()
201                 plabels_eval, loss_eval, _ = sess.run(
202                     [predicted_labels, loss, train_op],
203                     feed_dict={
204                         mlp._X: train_data,
205                         mlp._real_Y: train_labels
206                     }
207                 )
208                 step += 1
209                 print 'step: {}, loss: {}'.format(step, loss_eval)
```

> hàm load_dataset

```
170     def load_dataset():
171         train_data_reader = DataReader(
172             data_path='../datasets/20news-train-tfidf.txt',
173             batch_size=50,
174             vocab_size=vocab_size
175         )
176         test_data_reader = DataReader(
177             data_path='../datasets/20news-test-tfidf.txt',
178             batch_size=50,
179             vocab_size=vocab_size
180         )
181         return train_data_reader, test_data_reader
```

> class DataReader

```
121     class DataReader:
122         def __init__(self, data_path, batch_size, vocab_size):...
145
146         def next_batch(self):...
```

class DataReader: hàm init

```
121     class DataReader:
122         def __init__(self, data_path, batch_size, vocab_size):
123             self._batch_size = batch_size
124             with open(data_path) as f:
125                 d_lines = f.read().splitlines()
126
127             self._data = []
128             self._labels = []
129             for data_id, line in enumerate(d_lines):...
140
141             self._data = np.array(self._data)
142             self._labels = np.array(self._labels)
143
144             self._num_epoch = 0
145             self._batch_id = 0
```


class DataReader: hàm init:

```
129     for data_id, line in enumerate(d_lines):
130         vector = [0.0 for _ in range(vocab_size)]
131         features = line.split('<fff>')
132         label, doc_id = int(features[0]), int(features[1])
133         tokens = features[2].split()
134         for token in tokens:
135             index, value = int(token.split(':')[0]), \
136                             float(token.split(':')[1])
137             vector[index] = value
138         self._data.append(vector)
139         self._labels.append(label)
```

class DataReader: hàm next_batch

```
147     def next_batch(self):
148         start = self._batch_id * self._batch_size
149         end = start + self._batch_size
150         self._batch_id += 1
151
152         if end + self._batch_size > len(self._data):
153             end = len(self._data)
154             self._num_epoch += 1
155             self._batch_id = 0
156             indices = range(len(self._data))
157             random.seed(2018)
158             random.shuffle(indices)
159             self._data, self._labels = self._data[indices], self._labels[indices]
160
161         return self._data[start:end], self._labels[start:end]
```

Lưu các tham số mô hình: có thể lưu tại bất cứ bước lặp nào của quá trình training

```
trainable_variables = tf.trainable_variables()
for variable in trainable_variables:
    save_parameters(
        name=variable.name,
        value=variable.eval(),
        epoch=train_data_reader._num_epoch
    )
```

Lưu các tham số mô hình: hàm save_parameters

```
310 def save_parameters(name, value, epoch):
311     filename = name.replace(':', '-colon-') + '-epoch-{}.txt'.format(epoch)
312     if len(value.shape) == 1: # is a list
313         string_form = ','.join([str(number) for number in value])
314     else:
315         string_form = '\n'.join([''.join([str(number)
316                                         for number in value[row]])
317                                 for row in range(value.shape[0])])
318
319     with open('../saved-paras/' + filename, 'w') as f:
320         f.write(string_form)
```

Nội dung file

```
1 -0.450477,0.69281,1.19272,1.40951,-1.39532,-1.90461,0.46044,
2 1.96216,-1.09849,0.427538,0.810766,1.39989,0.338526,-2.33891
3 1.64311,-0.911746,0.330272,0.972807,-1.04616,1.89584,-1.4516
4 -1.76935,-3.79334,-4.09076,-1.2307,-2.96816,-1.92106,3.73486
5 -0.370901,0.237953,-0.345284,-3.19793,2.69816,-1.19778,-2.04
6 -0.405415,0.575845,0.929856,-3.26487,-3.37635,0.116976,-1.21
7 0.32717,-0.669439,-1.87453,2.23346,1.30141,2.47804,-2.98699,
8 -0.627788,-0.914528,2.59333,1.59767,3.11817,-3.31037,-0.7706
9 -0.36349,0.0887211,-0.52825,-3.55277,-2.00161,-1.63245,-2.76
```

Khôi phục các tham số đã lưu:

```
trainable_variables = tf.trainable_variables()
for variable in trainable_variables:
    saved_value = restore_parameters(variable.name, epoch)
    assign_op = variable.assign(saved_value)
    sess.run(assign_op)
```

Hàm restore_parameters

```
376 def restore_parameters(name, epoch):
377     filename = name.replace(':', '-colon-') + '-epoch-{}.txt'.format(epoch)
378     with open('../saved-paras/' + filename) as f:
379         lines = f.read().splitlines()
380         if len(lines) == 1: # is a vector
381             value = [float(number) for number in lines[0].split(',')]
382         else: # is a matrix
383             value = [[float(number) for number in lines[row].split(',')]
384                     for row in range(len(lines))]
385     return value
```

Đánh giá model trên test data:

```
335 test_data_reader = DataReader(  
336     data_path='../datasets/20news-test-tfidf.txt',  
337     batch_size=50,  
338     vocab_size=vocab_size  
339 )  
340 with tf.Session() as sess:  
341     epoch = 10  
342  
343     trainable_variables = tf.trainable_variables()  
344     for variable in trainable_variables:  
345         saved_value = restore_parameters(variable.name, epoch)  
346         assign_op = variable.assign(saved_value)  
347         sess.run(assign_op)  
  
349 num_true_preds = 0  
350 while True:  
351     test_data, test_labels = test_data_reader.next_batch()  
352     test_plabels_eval = sess.run(  
353         predicted_labels,  
354         feed_dict={  
355             mlp._X: test_data,  
356             mlp._real_Y: test_labels  
357         }  
358     )  
359     matches = np.equal(test_plabels_eval, test_labels)  
360     num_true_preds += np.sum(matches.astype(float))  
361  
362     if test_data_reader._batch_id == 0:  
363         break  
364 print 'Epoch:', epoch  
365 print 'Accuracy on test data:', num_true_preds / len(test_data_reader._data)
```

Tài liệu tham khảo

<http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

<http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>

[Tensorflow là gì? 10 tài liệu học tensorflow đầy đủ nhất | TopDev](#)

[Machine Learning cơ bản \(machinelearningcoban.com\)](#)