第十七章 潜在语义分析

1.单词向量空间模型通过单词的向量表示文本的语义内容。以单词-文本矩阵X为输入，其中每一行对应一个单词，每一列对应一个文本，每一个元素表示单词在文本中的频数或权值（如TF-IDF）

$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$ 单词向量空间模型认为，这个矩阵的每一列向量是单词向量，表示一个文本，两个单词向量的内积或标准化内积表示文本之间的语义相似度。

2.话题向量空间模型通过话题的向量表示文本的语义内容。假设有话题文本矩阵

$Y=\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & & \vdots \\ y_{k1} & y_{k2} & \cdots & y_{kn} \end{bmatrix}$ 其中每一行对应一个话题，每一列对应一个文本每一个元素表示话题在文本中的权值。话题向量空间模型认为，这个矩阵的每一列向量是话题向量，表示一个文本，两个话题向量的内积或标准化内积表示文本之间的语义相似度。假设有单词话题矩阵T

$T=\begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1k} \\ t_{21} & t_{22} & \cdots & t_{2k} \\ \vdots & \vdots & & \vdots \\ t_{m1} & t_{m2} & \cdots & t_{mk} \end{bmatrix}$ 其中每一行对应一个单词，每一列对应一个话题，每一个元素表示单词在话题中的权值。

给定一个单词文本矩阵X $X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$

潜在语义分析的目标是，找到合适的单词-话题矩阵T与话题文本矩阵Y，将单词文本矩阵X近似的表示为T与Y的乘积形式。 $X \approx TY$

等价地，潜在语义分析将文本在单词向量空间的表示X通过线性变换T转换为话题向量空间中的表示Y。

潜在语义分析的关键是对单词-文本矩阵进行以上的矩阵因子分解（话题分析）

3.潜在语义分析的算法是奇异值分解。通过对单词文本矩阵进行截断奇异值分解，得到
$X \approx U_k \Sigma_k V_k^T = U_k(\Sigma_k V_k^T)$

矩阵U_k表示话题空间，矩阵$(\Sigma_k V_k^T)$是文本在话题空间的表示。

4.非负矩阵分解也可以用于话题分析。非负矩阵分解将非负的单词文本矩阵近似分解成两个非负矩阵W和H的乘积，得到 $X \approx WH$

矩阵W表示话题空间，矩阵H是文本在话题空间的表示。

非负矩阵分解可以表为以下的最优化问题：$\min_{W,H} \|X-WH\|^2 \\ \text{s.t. } W,H \geq 0$ 非负矩阵分解的算法是迭代算法。乘法更新规则的迭代算法，交替地对W和H进行更新。本质是梯度下降法，通过定义特殊的步长和非负的初始值，保证迭代过程及结果的矩阵W和H均为非负。

LSA 是一种无监督学习方法，主要用于文本的话题分析，其特点是通过矩阵分解发现文本与单词之间的基于话题的语义关系。也称为潜在语义索引（Latent semantic indexing, LSI）。

LSA 使用的是非概率的话题分析模型。将文本集合表示为**单词-文本矩阵**，对单词-文本矩阵进行**奇异值分解**，从而得到话题向量空间，以及文本在话题向量空间的表示。

非负矩阵分解（non-negative matrix factorization, NMF）是另一种矩阵的因子分解方法，其特点是分解的矩阵非负。

单词向量空间

word vector space model

给定一个文本，用一个向量表示该文本的"语义"， 向量的**每一维对应一个单词**，其数值为该单词在该文本中出现的频数或权值；基本假设是文本中所有单词的出现情况表示了文本的语义内容，文本集合中的每个文本都表示为一个向量，存在于一个向量空间；向量空间的度量，如内积或标准化**内积**表示文本之间的**相似度**。

给定一个含有n个文本的集合$D=(d_1,d_2,\dots,d_n)$，以及在所有文本中出现的m个单词的集合$W=(w_1,w_2,\dots,w_m)$. 将单词在文本的出现的数据用一个单词-文本矩阵（word-document matrix）表示，记作X:

$$X=\begin{bmatrix} x_{11} & x_{12} & x_{1n} \\ x_{21} & x_{22} & x_{2n} \\ \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{mn} \end{bmatrix}$$

这是一个$m*n$矩阵，元素x_{ij}表示单词w_i在文本d_j中出现的频数或权值。由于单词的种类很多，而每个文本中出现单词的种类通常较少，所有单词-文本矩阵是一个稀疏矩阵。

权值通常用单词**频率-逆文本率**（term frequency-inverse document frequency, TF-IDF）表示：

$TF-IDF(t,d)=TF(t,d)*IDF(t),$

其中，$TF(t,d)$为单词t在文本d中出现的概率，$IDF(t)$是逆文本率，用来衡量单词t对表示语义所起的重要性，

$IDF(t)=log(\frac{len(D)}{len(t\in D)+1}).$

单词向量空间模型的优点是**模型简单，计算效率高**。因为单词向量通常是稀疏的，单词向量空间模型也有一定的局限性，体现在内积相似度未必能够准确表达两个文本的语义相似度上。因为自然语言的单词具有一词多义性（polysemy）及多词一义性（synonymy）。

话题向量空间

1. 话题向量空间：

给定一个含有n个文本的集合$D=(d_1,d_2,\ldots,d_n)$，以及在所有文本中出现的m个单词的集合$W=(w_1,w_2,\ldots,w_m)$. 可以获得其单词-文本矩阵X:

$$X=\begin{bmatrix} x_{11} & x_{12} & x_{1n} \\ x_{21} & x_{22} & x_{2n} \\ \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{mn} \end{bmatrix}$$

假设所有文本共含有k个话题。假设每个话题由一个定义在单词集合W上的m维向量表示，称为话题向量，即：

$$t_l=\begin{bmatrix} t_{1l} \\ t_{2l} \\ \vdots \\ t_{ml} \end{bmatrix},l=1,2,\ldots,k$$

其中t_{il}单词w_i在话题t_l的权值，$i=1,2,\ldots,m$, 权值越大，该单词在该话题中的重要程度就越高。这k个话题向量 $t_1,t_2,\ldots,t_k$张成一个话题向量空间（topic vector space）， 维数为k.**话题向量空间是单词向量空间的一个子空间**。

话题向量空间T：

$$T=\begin{bmatrix} t_{11} & t_{12} & t_{1k} \\ t_{21} & t_{22} & t_{2k} \\ \vdots & \vdots & \vdots \\ t_{m1} & t_{m2} & t_{mk} \end{bmatrix}$$

矩阵T,称为**单词-话题矩阵**。$T=[t_1,t_2,\ldots,t_k]$

2. 文本在话题向量空间中的表示：

考虑文本集合D的文本d_j, 在单词向量空间中由一个向量x_j表示，将x_j投影到话题向量空间T中，得到话题向量空间的一个向量y_j, y_j是一个k维向量：

$$y_j=\begin{bmatrix} y_{1j} \\ y_{2j} \\ \vdots \\ y_{kj} \end{bmatrix},j=1,2,\ldots,n$$

其中，y_{lj}是文本d_j在话题t_l中的权值，$l=1,2,\ldots,k$, 权值越大，该话题在该文本中的重要程度就越高。

矩阵Y 表示话题在文本中出现的情况，称为话题-文本矩阵（topic-document matrix）,记作：

$$Y=\begin{bmatrix} y_{11} & y_{12} & y_{1n} \\ y_{21} & y_{22} & y_{2n} \\ \vdots & \vdots & \vdots \\ y_{k1} & y_{k2} & y_{kn} \end{bmatrix}$$

也可写成：$Y=[y_1,y_2\ldots,y_n]$

3. 从单词向量空间到话题向量空间的线性变换：

如此，单词向量空间的文本向量x_j可以通过他在话题空间中的向量y_j近似表示，具体地由k个话题向量以y_j为系数的线性组合近似表示：

$$x_j = y_{1j}t_1 + y_{2j}t_2 + \ldots + y_{yj}t_k, j = 1, 2, \ldots, n$$

所以，单词-文本矩阵X可以近似的表示为单词-话题矩阵T与话题-文本矩阵Y的乘积形式。

$$X \approx TY$$

直观上，潜在语义分析是将单词向量空间的表示通过线性变换转换为在话题向量空间中的表示。这个线性变换由矩阵因子分解式的形式体现。

潜在语义分析算法

潜在语义分析利用矩阵奇异值分解，具体地，对单词-文本矩阵进行奇异值分解，将其左矩阵作为话题向量空间，将其对角矩阵与右矩阵的乘积作为文本在话题向量空间的表示。

给定一个含有n个文本的集合$D = (d_1, d_2, \ldots, d_n)$，以及在所有文本中出现的$m$个单词的集合$W = (w_1, w_2, \ldots, w_m)$. 可以获得其单词-文本矩阵$X$:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{1n} \\ x_{21} & x_{22} & x_{2n} \\ \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{mn} \end{bmatrix}$$

截断奇异值分解：

潜在语义分析根据确定的话题数k对单词-文本矩阵X进行截断奇异值分解：

$$X \approx U_k \Sigma_k V_k^T = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ & & \ddots & \\ 0 & 0 & & 0 \\ 0 & 0 & 0 & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \end{bmatrix}$$

矩阵U_k的每一个列向量 $u_1, u_2, \ldots, u_k$ 表示一个话题，称为**话题向量**。由这 k 个话题向量张成一个子空间：

$$U_k = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix}$$

称为**话题向量空间**。

综上， 可以通过对单词-文本矩阵的奇异值分解进行潜在语义分析：

$$X \approx U_k \Sigma_k V_k^T = U_k (\Sigma_k V_k^T)$$

得到话题空间 U_k ， 以及文本在话题空间的表示($\Sigma_k V_k^T$).

非负矩阵分解算法

非负矩阵分解也可以用于话题分析。对单词-文本矩阵进行非负矩阵分解，将**其左矩阵作为话题向量空间**，将其**右矩阵作为文本在话题向量空间的表示**。

非负矩阵分解

若一个矩阵的索引元素非负，则该矩阵为非负矩阵。若X是非负矩阵，则： $X >= 0$.

给定一个非负矩阵X, 找到两个非负矩阵$W >= 0$ 和 $H >= 0$, 使得：

$$X \approx WH$$

即非负矩阵X分解为两个非负矩阵W和H的乘积形式，成为非负矩阵分解。因为WH与X完全相等很难实现，所以只要求近似相等。

假设非负矩阵X是$m \times n$矩阵，非负矩阵W和H分别为 $m \times k$ 矩阵和 $k \times n$ 矩阵。假设 $k < min(m, n)$ 即W 和 H 小于原矩阵 X, 所以非负矩阵分解是对原数据的压缩。

称 W 为基矩阵， H 为系数矩阵。非负矩阵分解旨在用较少的基向量，系数向量来表示为较大的数据矩阵。

令 $W = [\, w_1 \quad w_2 \quad \cdots \quad w_k \,]$ 为话题向量空间， $w_1, w_2, \ldots, w_k$ 表示文本集合的 k 个话题， 令 $H = [\, h_1 \quad h_2 \quad \cdots \quad h_n \,]$ 为文本在话题向量空间的表示， $h_1, h_2, \ldots, h_n$ 表示文本集合的 n 个文本。

算法

非负矩阵分解可以形式化为最优化问题求解。可以利用平方损失或散度来作为损失函数。

目标函数 $||X - WH||^2$ 关于 W 和 H 的最小化，满足约束条件 $W, H >= 0$, 即：

$$\min_{W,H} ||X - WH||^2$$

$$s.t. W, H >= 0$$

乘法更新规则：

$$W_{il} \leftarrow W_{il} \frac{(XH^T)_{il}}{(WHH^T)_{il}} \quad (17.33)$$

$$H_{lj} \leftarrow H_{lj} \frac{(W^T X)_{lj}}{(W^T WH)_{lj}} \quad (17.34)$$

选择初始矩阵 W 和 H 为非负矩阵，可以保证迭代过程及结果的矩阵 W 和 H 非负。

算法 17.1 （非负矩阵分解的迭代算法）

输入： 单词-文本矩阵 $X >= 0$, 文本集合的话题个数 k, 最大迭代次数 t；
输出： 话题矩阵 W, 文本表示矩阵 H。

1). 初始化

$W >= 0$, 并对 W 的每一列数据归一化；
$H >= 0$；

2). 迭代

对迭代次数由1到t执行下列步骤：
a. 更新W的元素，对 l 从1到 k, i从1到m按(17.33)更新 W_{il};
a. 更新H的元素，对 l 从1到 k, j从1到m按(17.34)更新 H_{lj};

图例 17.1

```
import numpy as np
from sklearn.decomposition import TruncatedSVD
```

```
X = [[2, 0, 0, 0], [0, 2, 0, 0], [0, 0, 1, 0], [0, 0, 2, 3], [0, 0, 0, 1], [1, 2, 2, 1]]
X = np.asarray(X);X
```

```
array([[2, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 2, 3],
       [0, 0, 0, 1],
       [1, 2, 2, 1]])
```

```
# 奇异值分解
U,sigma,VT=np.linalg.svd(X)
```

```
U
```

```
array([[-7.84368672e-02, -2.84423033e-01,  8.94427191e-01,
        -2.15138396e-01, -6.45002451e-02, -2.50012770e-01],
       [-1.56873734e-01, -5.68846066e-01, -4.47213595e-01,
        -4.30276793e-01, -1.29000490e-01, -5.00025540e-01],
       [-1.42622354e-01,  1.37930417e-02, -1.25029761e-16,
         6.53519444e-01,  3.88575115e-01, -6.33553733e-01],
       [-7.28804669e-01,  5.53499910e-01, -2.24565656e-16,
        -1.56161345e-01, -3.23288048e-01, -1.83248673e-01],
       [-1.47853320e-01,  1.75304609e-01,  8.49795536e-18,
        -4.87733411e-01,  8.40863653e-01,  4.97204799e-02],
       [-6.29190197e-01, -5.08166890e-01, -1.60733896e-16,
         2.81459486e-01,  1.29000490e-01,  5.00025540e-01]])
```

```
sigma
```

```
array([4.47696617, 2.7519661 , 2.        , 1.17620428])
```

```
VT
```

```
array([[-1.75579600e-01, -3.51159201e-01, -6.38515454e-01,
        -6.61934313e-01],
       [-3.91361272e-01, -7.82722545e-01,  3.79579831e-02,
         4.82432341e-01],
       [ 8.94427191e-01, -4.47213595e-01, -2.23998094e-16,
         5.45344065e-17],
       [-1.26523351e-01, -2.53046702e-01,  7.68672366e-01,
        -5.73674125e-01]])
```

```
# 截断奇异值分解

svd = TruncatedSVD(n_components=3, n_iter=7, random_state=42)
svd.fit(X)
```

```
TruncatedSVD(algorithm='randomized', n_components=3, n_iter=7,
random_state=42,
             tol=0.0)
```

```
print(svd.explained_variance_ratio_)
```

```
[0.39945801 0.34585056 0.18861789]
```

```
print(svd.explained_variance_ratio_.sum())
```

```
0.933926460028446
```

```
print(svd.singular_values_)
```

```
[4.47696617 2.7519661  2.        ]
```

非负矩阵分解

```
def inverse_transform(W, H):
    # 重构
    return W.dot(H)

def loss(X, X_):
    #计算重构误差
    return ((X - X_) * (X - X_)).sum()
```

```
# 算法 17.1

class MyNMF:
    def fit(self, X, k, t):
        m, n = X.shape

        W = np.random.rand(m, k)
        W = W/W.sum(axis=0)

        H = np.random.rand(k, n)

        i = 1
        while i < t:

            W = W * X.dot(H.T) / W.dot(H).dot(H.T)

            H = H * (W.T).dot(X) / (W.T).dot(W).dot(H)

            i += 1

        return W, H
```

```
model = MyNMF()
W, H = model.fit(X, 3, 200)
```

```
W
```

```
array([[0.00000000e+00, 4.27327705e-01, 6.30117924e-27],
       [5.11680721e-97, 8.57828102e-01, 0.00000000e+00],
       [2.97520805e-88, 2.39454414e-18, 4.36332453e-01],
       [2.15653741e+00, 3.38756557e-21, 8.38350315e-01],
       [7.36106818e-01, 1.00339294e-54, 8.72892573e-38],
       [6.78344810e-01, 1.07009504e+00, 8.57259947e-01]])
```

```
H
```

```
array([[7.14647214e-10, 1.01233436e-03, 3.76097657e-02, 1.35755597e+00],
       [9.30509415e-01, 1.86788842e+00, 1.16682319e-02, 4.54479182e-03],
       [4.95440453e-03, 6.18432747e-04, 2.28890170e+00, 8.61836630e-02]])
```

```
# 重构
X_ = inverse_transform(W, H);X_
```

```
array([[3.97632453e-01, 7.98200474e-01, 4.98615876e-03, 1.94211546e-03],
       [7.98217125e-01, 1.60232718e+00, 1.00093372e-02, 3.89865014e-03],
       [2.16176748e-03, 2.69842277e-04, 9.98722093e-01, 3.76047290e-02],
       [4.15352814e-03, 2.70160021e-03, 2.00000833e+00, 2.99987233e+00],
       [5.26056687e-10, 7.45186228e-04, 2.76848049e-02, 9.99306203e-01],
       [9.99980721e-01, 2.00003500e+00, 2.00018226e+00, 9.99636205e-01]])
```

```
# 重构误差

loss(X, X_)
```

```
4.002356735601103
```

使用 sklearn 计算

```
from sklearn.decomposition import NMF
model = NMF(n_components=3, init='random', max_iter=200, random_state=0)
W = model.fit_transform(X)
H = model.components_
```

```
W
```

```
array([[0.        , 0.53849498, 0.        ],
       [0.        , 1.07698996, 0.        ],
       [0.69891361, 0.        , 0.        ],
       [1.39782972, 0.        , 1.97173859],
       [0.        , 0.        , 0.65783848],
       [1.39783002, 1.34623756, 0.65573258]])
```

```
H
```

```
array([[0.00000000e+00, 0.00000000e+00, 1.43078959e+00, 1.71761682e-03],
       [7.42810976e-01, 1.48562195e+00, 0.00000000e+00, 3.30264644e-04],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.52030365e+00]])
```

```
X__ = inverse_transform(W, H);X__
```

```
array([[3.99999983e-01, 7.99999966e-01, 0.00000000e+00, 1.77845853e-04],
       [7.99999966e-01, 1.59999993e+00, 0.00000000e+00, 3.55691707e-04],
       [0.00000000e+00, 0.00000000e+00, 9.99998311e-01, 1.20046577e-03],
       [0.00000000e+00, 0.00000000e+00, 2.00000021e+00, 3.00004230e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00011424e+00],
       [1.00000003e+00, 2.00000007e+00, 2.00000064e+00, 9.99758185e-01]])
```

```
loss(X, X__)
```

```
4.000001672582457
```

本章代码来源：https://github.com/hktxt/Learn-Statistical-Learning-Method

本文代码更新地址：https://github.com/fengdu78/lihang-code

中文注释制作：机器学习初学者公众号：ID:ai-start-com

配置环境：python 3.5+

代码全部测试通过。