IIIS 2018 Spring: ATCS - Selected Topics in Optimization
Lecture date: April 9th, 2018
Instructor: Jian Li **Scribe: Tianzong Zhang, Yao Class 60**

1 Topic Modeling

1.1 Nonnegative matrix factorization

On the recitation TA Zhang has taught SVD and PCA. In this algorithm, the main idea is to factorize $A = U^T \Sigma V$, where U, V are unitary. We can show that A and Σ have the same rank.

In this lecture, we consider another class of factorization, called **nonnegative matrix factorization**. [1][2][3] This technique is widely used in NLP. The problem is stated as follows:

Definition 1 (Nonnegative matrix factorization (NMF)) *Suppose we have a $m \times n$ matrix M (probably low rank), with all entries nonnegative and each column sum be 1. The goal is to factorize*

$$M = AW,$$

where all entries of A, W are also nonnegative, and each column has sum 1. Suppose A is $m \times r$, W is $r \times n$, we want $r \ll n$.

The image is like this:

$$\begin{bmatrix} \\ M \\ \\ \end{bmatrix} = \begin{bmatrix} \\ A \\ \\ \end{bmatrix} \begin{bmatrix} \quad W \quad \end{bmatrix}.$$

We briefly introduce the application of NMF. Suppose M is a matrix shows the word frequency of words in different documents. Let n is number of English words, m is the number of a documents. M_{ij} is the probability of word i appearing in document j. Each document have a special probability distribution of words. For example, *football* have more probability than *president* in a sport document. If we can factorize M, then each column of A is a topic and each column of W is a topic distribution. The following Figure 1 intuitively shows how this works.

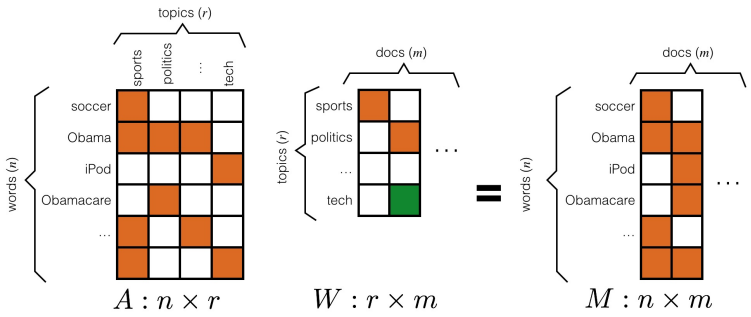

Figure 1: The figure of the relationship between M and A, W

This is a generative model for documents. Think each document as a probability distribution (convex combination) of topics. Each topic is a probability distribution (convex combination) of words. Then a document is simply generated by repeat L times the following process:
(1) pick a topic $t \sim$ distr(topics),
(2) pick a word $w \sim$ distr(words for topic t) .

This problem is actually hard. We have the following theorem (without proof):

Theorem 2 *NMF is NP-hard in general.*

However, we can try to solve the problem in some special cases. Here we introduce the concept of separable.

Definition 3 *The matrix A is separable, if each column has an anchor word, i.e. this anchor word only appears in this topic, i.e. the probability of this word is positive only in this topic. As an example, an image is as follows. In this example, word i is the anchor word of topic j.*

$$A = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & \cdots & j & \cdots & r \end{array} \\ \begin{pmatrix} & & & & & \\ & & & & & \\ & & & & & \\ 0 & 0 & \cdots & 0.00001 & \cdots & 0 \\ & & & & & \\ & & & & & \end{pmatrix} \end{array},$$

Under the assumption of separable, we can deploy an algorithm for NMF. The algorithm is as follows.

First we rescale M, A, W s.t. each row sum is 1. (In fact, this can be done, since if each row sum of M, W is 1, then since each row of M is the linear combination of each row of W with coefficient in A, hence each row sum of A is 1 as well. Check it by yourself). Therefore, there is a square submatrix of A, and the submatrix is a identity matrix.

We have the following observations.

Fact 1 *We have an identity matrix of size r in matrix A. The corresponding submatrix in M is just W. Moreover, each other row of M is a convex combination of rows of W.*

A more intuitive image of this observation, please refer to Figure 2.

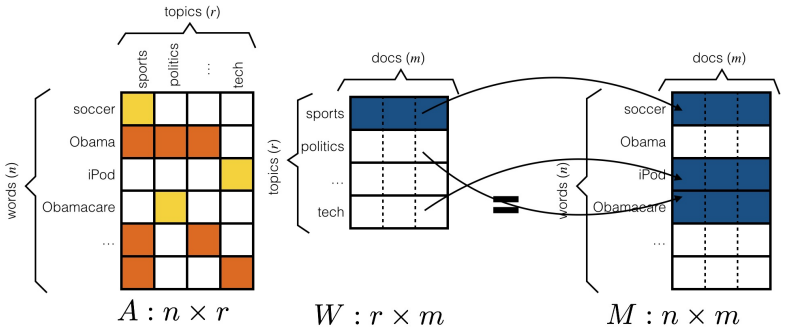

Figure 2: An intuitive understanding of anchor word and the observation

There is an geometry explanation of the observation. We consider each row of M be a point in a n-dimensional space. Then all rows of $W \subseteq M$ form r vertices of a simplex in the space, and all other rows in $M \backslash W$ is inside the simplex. A simple example is illustrated in Figure 3.

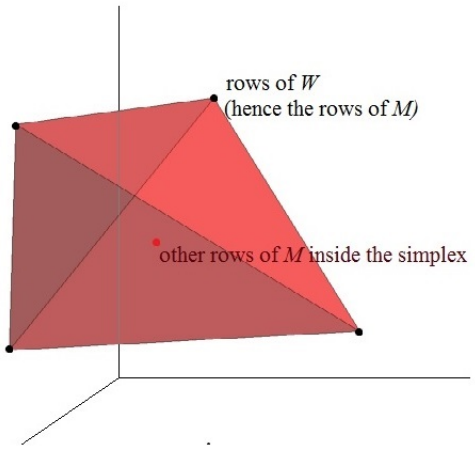

Figure 3: The geometry meaning of anchor words

According to the observation, we can directly find the rows which will form matrix W in M. We deploy the following algorithm.

Algorithm 1: Recover anchor words by using convex hull method

Initially, set $I = [m]$;
for $i = 1, 2, \cdots, m$ **do**
  if $M_i \in \text{ConvCombine}\{M_j | j \in I\}$ **then**
    $I \leftarrow I - \{i\}$;
Return I = set of row indices corresponding to anchor words;

The only remaining problem is, how to check whether M_i is in the ConvCombine$\{M_j | j \in I\}$?

This problem is equivalent to solving the following linear programming problem:

$$\begin{aligned} \text{(LP)} \quad M_i &= \sum_{j \in I, j \neq i} \alpha_j M_j \\ \sum_{j \in I, j \neq i} \alpha_j &= 1 \\ a_j &\geq 0, \forall j \in I, j \neq i. \end{aligned}$$

Now we know that $W = M_I$ (the submatrix of M), hence solving A is equivalent to the least square problem: to minimize

$$||M - AW||_F,$$

which one should expect to get 0. Recall that F is the Frobenius norm, and

$$||M||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |M_{ij}|^2}.$$

Now we are done.

1.2 Topic Modeling for Large Sample with Few Words

Sometimes anchor word won't appear. Consider the following scenario: the number of document is huge but the length of each document is small, for example, Twitter. Then the matrix M is of great noise. Now we denote the word-word co-occurrence matrix $Q = MM^T$. In fact, since sample is large, hence Q is of small noise, i.e. it is close to the expectation.

Under this situation, we want to factorize Q instead of M. Then

$$Q = MM^T = \mathbb{E}[AWW^T A^T] = A\mathbb{E}[WW^T]A^T = ARA^T,$$

where $R = \mathbb{E}[WW^T]$. Similarly we can identify the anchor word. The process is as follows: without rescaling (but with permuting), we have

$$Q = ARA^T = \begin{bmatrix} D \\ U \end{bmatrix} R \begin{bmatrix} D & U^T \end{bmatrix} = \begin{bmatrix} DRD & DRU^T \\ URD & URU^T \end{bmatrix},$$

where D is diagonal matrix which corresponds to the anchor words.

We recover A using the following 4 steps:

Algorithm 2: Recover A and R by factorizing word-word co-occurrence matrix Q

1 Calculate $\boldsymbol{p}_I \leftarrow Q_I \mathbf{1}$, where I is the rows index of anchor word. ;
2 Solve $\boldsymbol{z}$ where $Q_{I,I}\boldsymbol{z} = \boldsymbol{p}_I$;
3 Calculate $A^T \leftarrow (Q_{I,I}\text{diag}(\boldsymbol{z}))^{-1}Q_I$,;
4 $R = \text{diag}(\boldsymbol{z})Q_{I,I}\text{Diag}(\boldsymbol{z})$.

We prove the correctness of the algorithm.

Proof: According to Line 1, we have (since A is not rescaled)

$$\boldsymbol{p}_I = DRA^T\mathbf{1} = DR\mathbf{1},$$

and due to line 2,

$$DRD\boldsymbol{z} = DR\mathbf{1} \Rightarrow \text{diag}(\boldsymbol{z}) = D^{-1}.$$

Line 3 calculates the right A since

$$(Q_{I,I}\text{diag}(\boldsymbol{z}))^{-1}Q_I^T = (DRDD^{-1})^{-1}DRA^T = A^T.$$

And Line 4 finally outputs the right R, which easy to show since $DRD = Q_{I,I}$, which completes the proof. □

However, one issue is that we have inverse calculation in (3), which is sensitive to noise. There is an improved algorithm, but it will not be covered in class and please refer to [3].

1.3 Collective Filtering

We will discuss this problem for short. Consider you're operating an *eBay* website. There is a preference matrix, where each row is a user and each column is an item. For each user row, there is some real number that shows his preference to the items.

However, for each user there would be many missing entries because he may not see this item before. We want to recommend the item to the user.

$$A = \begin{array}{c} \\ \text{user 1} \\ \text{user 2} \\ \text{user 3} \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} \text{item 1} & \text{item 2} & \text{item 3} & \text{item 4} & \cdots \end{array} \\ \begin{pmatrix} 1 & ? & -1 & ? & \cdots \\ 0 & -1 & ? & 1 & \cdots \\ ? & ? & ? & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array},$$

This introduce the problem of **matrix completion**. We introduce the problem but will not go into detail.

Definition 4 (Matrix Completion) *Given a matrix M with some missing entries, we want to fill in the blanks of M and minimize rank(M), such that $M_{ij} = A_{ij}$ for observed values. Note that matrix rank is 0-norm, we can relax the problem to the nuclear norm $||M||_{nuclear}$.*

Note: nuclear norm is the sum of all the singular values of M, it is convex. Think the two following problems by yourself:

- Compare 0-norm(rank) with nuclear norm, why it is a relaxation?
- What is the dual norm of nuclear norm?

2 Word embedding

2.1 Word2Vec

We first introduce the definition of co-occurrence matrix. The co-occurrence matrix is square matrix, and column i and row i means some word. Given a corpus (a set of sentences), the entry of the matrix is the number of sentences that contain both the column word and the row word.

For example, the corpus is {"I like deep learning.", "I like NLP.", "I enjoy flying."}, then the co-occurrence matrix is as Fig. 4.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Figure 4: Co-occurrence matrix in the example above

The co-occurrence matrix is of high dimension and sparse. We want a relatively low-dimensional representation. We seek to use vector for representing each words.

Problem: For each word, we want to map it to a vector, which means some characteristic of the word.

One naive thought is to use SVD/PCA to deduce dimension. Instead of counting, we can use Pearson correlation when implementing SVD/PCA. There are several interesting semantic patterns in the vectors, as the following Fig. 5 show.

However, the computational cost of SVD scales quadratically, which is expensive for large vocabulary. Moreover, it is hard to incorporate new words or documents, and the algorithm is special and doesn't fit into DL pipeline. We have to find another way to solve the problem.

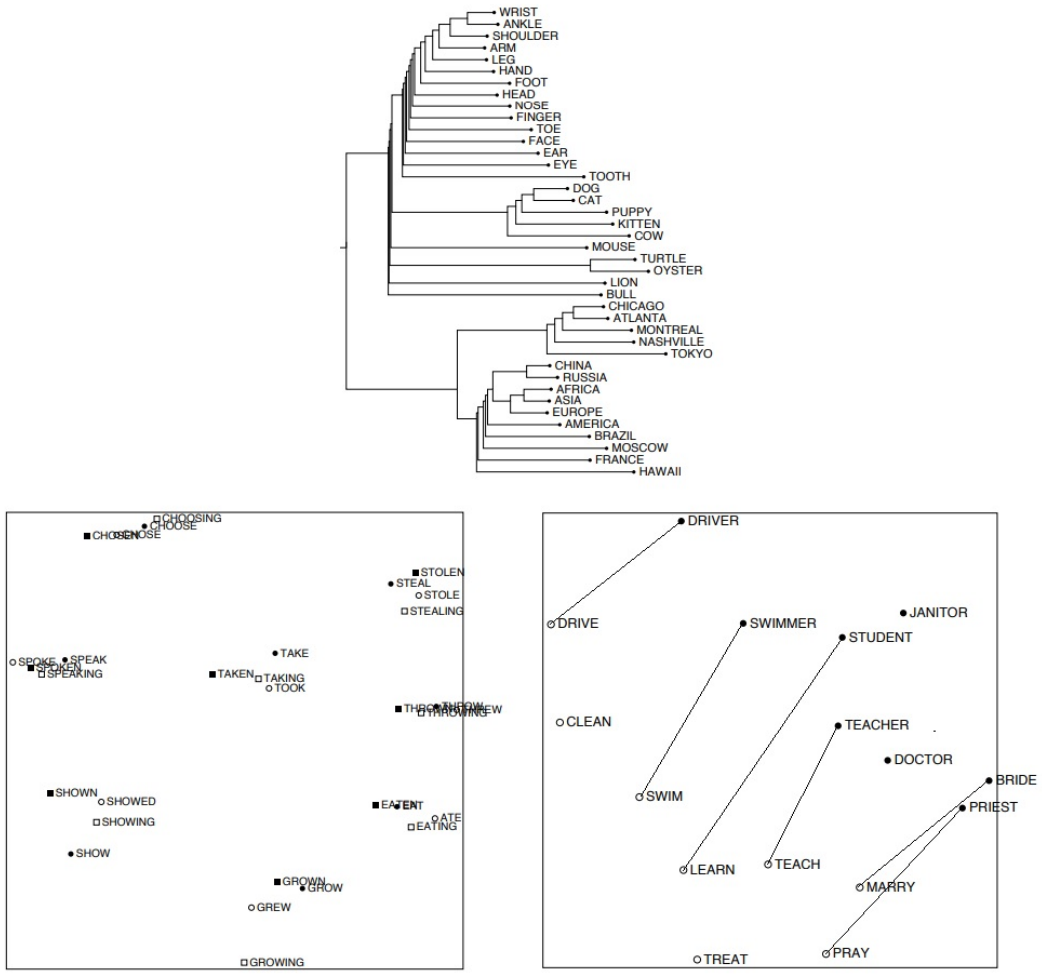

Figure 5: interesting semantic patterns in the vectors, for reference see [4]

We then introduce the technique of Word2Vec, which is the work by Mikolov, et. al. [5]. Instead of capturing co-occurrence counts directly, Word2Vec predicts surrounding words in a m-size window of every round. The details of Word2Vec is as follows.

Word2vec is a kind of unsupervised learning. The objective function of Word2Vec is to maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t),$$

where θ represents all variables we optimize.

The first $\sum$ notation means the sum over all words, and the second $\sum$ notation means the log probability of surrounding words of word w_t. This probability model is defined as follows:

$$p(o|c) = \frac{\exp(\boldsymbol{u}_o^T \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^T \boldsymbol{v}_c)},$$

where o is the outside word index, c is the center word index, $\boldsymbol{u}$ and $\boldsymbol{v}$ are center and outside vectors of o and c. This is a kind of dynamic logistic regression, and if $\boldsymbol{u}_o$ and $\boldsymbol{v}_c$ are with similar direction, then the probability is large. Notice that each word w have two vectors representing it, $\boldsymbol{u}_w$ means the embedding of w when w is outside, and $\boldsymbol{v}_w$ means the embedding of w when w is the center.

Training $\nabla \log p(w_o|w_c)$ takes $O(W)$ time and is expensive. There are two methods for training Word2Vec.

The first method is called **hierarchical softmax** method. In this method, we encode all the words in a Hoffman tree, see Fig. 6.

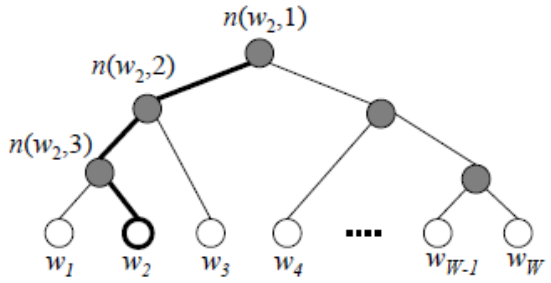

Figure 6: Huffman tree in hierarchical softmax method in Word2Vec

We denote $n(w, j)$ be the j-th node on the path from root to leaf w, and $L(w)$ be the length of the path. We calculate probability $p(w|w_c)$ as follows:

$$p(w|w_c) = \sum_{j=1}^{L(w)-1} \delta(\mathbb{I}\{n(w, j+1) = \text{lch}(n(w, j))\}\boldsymbol{u}'^T_{n(w,j)}\boldsymbol{v}_{w_c}),$$

where $\delta(\cdot)$ is the softmax function, $\mathbb{I}\{\cdot\}$ is the indicator function, $\text{lch}(\cdot)$ is the left child node function. One can that

$$\sum_{w=1}^{W} p(w|w_c) = 1.$$

This is actually an approximation of the previous probabilistic model, and the running time has decreased to the depth of the tree, i.e. $O(\log W)$ rather than $O(W)$. From this method we can further calculate gradient etc. [6]

The other training method is called **negative sampling** (NEG) method. Gutmann and Hyvarinen [7] introduced the technique of noise contrastive estimation (NCE). NCE posits that a good model should be able to differentiate data from noise (by logistic regression). NCE can be shown to approximately maximize the log probability of the softmax, and we can simplify it to the negative sampling method.

The negative sampling is to use the following objective (typically k is set $k \in [5, 20]$, and if data is large, we set $k \in [2, 5]$)

$$\log \delta(\boldsymbol{u}^T_{w_o}\boldsymbol{v}_{w_c}) + \sum_{i=1}^{k} \delta(-\boldsymbol{u}^T_{w_i}\boldsymbol{v}_{w_c})$$

to replace the $\log p(w_o|w_c)$ term in the objective function of Word2Vec:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t),$$

Thus the task is to distinguish the target word w_o from draws from the noise distribution $P(w)$ using logistic regression, where there are k negative samples for each data sample. For the free parameter, noise distribution $p(w)$, it can be set uniform distribution, unigram distribution $U(w)$ (i.e. according to word frequency), or $U(w)^{3/4}/Z$.

In the next class, we will look into some theoretical interpretation of the methods of Word2Vec.

References

[1] DD Lee. Algorithms for Non-negative Matrix Factorization. 2000.

[2] Catalin Voss. Building Topic Models Based on Anchor Words (Lecture Note). CS167, Stanford, 2014.

[3] Sanjeev Arora, Rong Ge, Yoni Halpern, David Mimno, Ankur Moitra, David Sontag, Yichen Wu, Michael Zhu. A Practical Algorithm for Topic Modeling with Provable Guarantees. 2012.

[4] Rohde. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence. 2005.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013.

[6] Xin Rong. Word2vec Parameter Learning Explained. 2016.

[7] Michael U Gutmann, Aapo Hyvarinen. Noise-contrastive Estimation of Unnormalized Statistical Models, With Applications to Natural Image Statistics. 2012.