Various Unsupervised learning models

Richard Xu

August 15, 2022

1 Introduction

This note describes some of the common topics in unsupervised learning. From the most obvious methods like clustering, to topic modelling (Latent Diricher Allocation) and to traditional word embedding, such as word2vec algorithm.

The purpose of this note is to provide intuitive explanations for all students, with or without a math background, so they may not be as rigorous. I also removed the integral and matrix-vector representations for the entire note. I used summation and scalar for this note. Therefore, there are little linear algebra nor calculus requirements to understand this note.

2 clustering

One of the most common unsupervised learning tasks is clustering. In the past, we have studied modeling data $\mathbf{x}$ with the label y. However, for the remaining studies, we will look at all data mining techniques for unlabeled data.

3 K-means clustering

It's been around for more than 40 years and it is still as effective as ever. Let me demonstrate how the algorithm works using the following set of data points:

$$\begin{array}{l}(-2, 1.5)\\(-1, 1)\\(-2, 3)\\(-1, 2.5)\\(-0.5, 3)\\(-2, -1.8)\\(-1, -1.5)\\(2, -1.5)\\(1, -1)\\(2, -3)\\(1, -2.5)\\(1, -3)\end{array} \tag{1}$$

These data points are plotted in gray color.

after deciding there will be two cluster, i.e., $K = 2$. By the way, this is usually the hardest choice of any clustering algorithm.

1. **STEP 1 initialization:** Choose the initial K centroids
   In our demonstration example, we have chosen a "very bad" centroids:

$$(-2.5, 1.5), (-1.5, 1.0) \tag{2}$$

   they are represented using red colour.

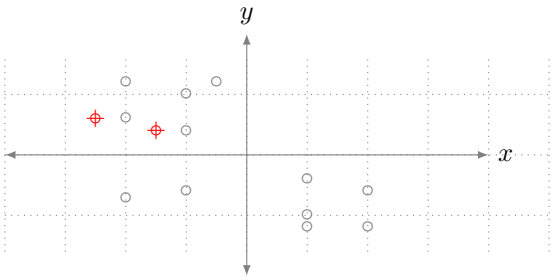


Figure 1: **STEP 1**: K-means at initialization

2. **STEP 2 Cluster assignment:** Assign each object to the group that has the closest centroid.

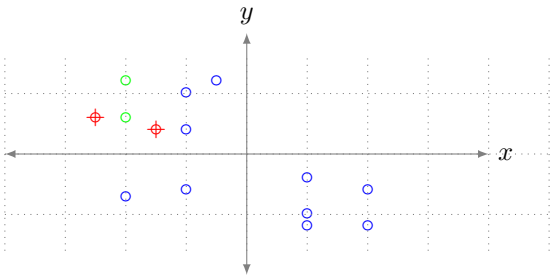


Figure 2: **STEP 2**: Cluster assignment iteration 1

in this step, the data points are assigned to the closest centroid. You can see that:

(a) $(-2, 1.5), (-2, 3)$ (represented in green colour) are associated with the centroid $(-2.5, 1.5)$
(b) $(-2, -1.8), (-1, -1.5), (2, -1.5), (1, -1), (2, -3), (1, -2.5), (1, -3), (-1, 1), (-1, 2.5), (-0.5, 3)$ (represented in blue colour) are associated with the centroid $(-1.5, 1.0)$

Expressing this association mathematically, for each cluster i, there are a set of points $S_i^{(t)}$ that are closest to centroid μ_i (meaning it's closer to $\mu_{j\neq i}$ than any other centroids:

$$S_l = \left\{ \mathbf{x}_i : \left\| \mathbf{x}_i - \mu_l \right\|^2 \leq \left\| \mathbf{x}_i - \mu_m \right\|^2 \quad \forall m : 1 \leq m \leq K \right\} \tag{3}$$

since the cluster assignment S_i changes each iteration, so we also put the iteration index g, in here $g = 1$:

$$S_l^{(g)} = \left\{ \mathbf{x}_i : \left\| \mathbf{x}_i - \mu_l^{(g)} \right\|^2 \leq \left\| \mathbf{x}_i - \mu_m^{(g)} \right\|^2 \quad \forall m : 1 \leq m \leq K \right\} \tag{4}$$

3. **STEP 3 cluster centroid update:** When all objects have been assigned, recalculate the positions of the K centroids.

   numerically, because:

   (a) $(-2, 1.5), (-2, 3)$ (represented in green colour) are associated with the centroid $(-2.5, 1.5)$, then:

   $$\frac{(-2, 1.5) + (-2, 3)}{2} = \frac{(-4, 4.5)}{2} = (-2, 2.25) \tag{5}$$

   (b) $(-2, -1.8), (-1, -1.5), (2, -1.5), (1, -1), (2, -3), (1, -2.5), (1, -3), (-1, 1), (-1, 2.5), (-0.5, 3)$ (represented in blue colour) are associated with the centroid $(-1.5, 1.0)$, then:

   $$\begin{aligned} &\frac{(-2, -1.8) + (-1, -1.5) + (2, -1.5) + (1, -1) + (2, -3) + (1, -2.5) + (1, -3) + (-1, 1) + (-1, 2.5) + (-0.5, 3)}{10} \\ &= \frac{(1.5, -7.8)}{10} = (0.15, -0.78) \end{aligned} \tag{6}$$

   note we have vector operation where $(a_1 + b_1) + (a_2 + b_2) = (a_1 + a_2, b_1 + b_2)$

   Therefore, the centroids was changed from the previous iteration:

   $$\begin{aligned} (-2.5, 1.5) &\rightarrow (-2, 2.25) \\ (-1.5, 1.0) &\rightarrow (0.15, -0.78) \end{aligned} \tag{7}$$

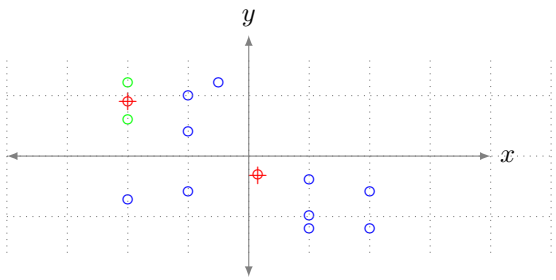


Figure 3: **STEP 3**: cluster centroid update iteration 1

Expressing it mathematically:

$$\mu_i^{(g+1)} = \frac{1}{\left| S_i^{(g)} \right|} \sum_{\mathbf{x}_j \in S_i^{(g)}} \mathbf{x}_j \tag{8}$$

3.1 repeat Cluster assignment and cluster centroid update until convergence

We need to repeat Steps 2 (Cluster assignment) and 3 (cluster centroid update) until the centroids no longer move, i.e., the entire algorithm converge.

I omitted the numerical computation in this section, as the computation are exactly the same as in Iteration 1. Also note that for all the following diagrams, I show the results using eyeball estimation. They may not be exact.

3.1.1 2^{nd} iteration

Cluster assignment

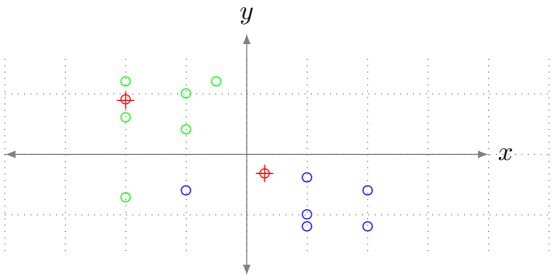


Figure 4: cluster assignment at iteration 2

cluster centroid update

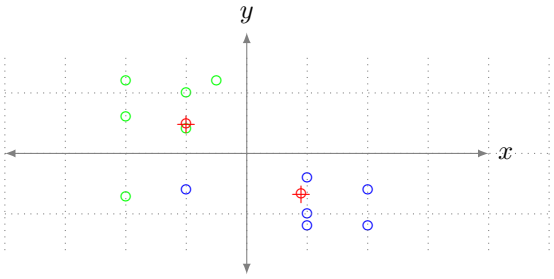


Figure 5: cluster centroid update in iteration 2

3.2 3^{rd} iteration

Cluster assignment

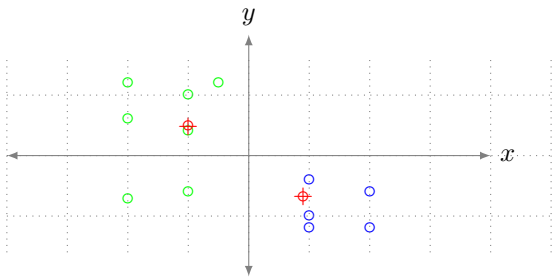


Figure 6: cluster assignment in iteration 3

cluster centroid update

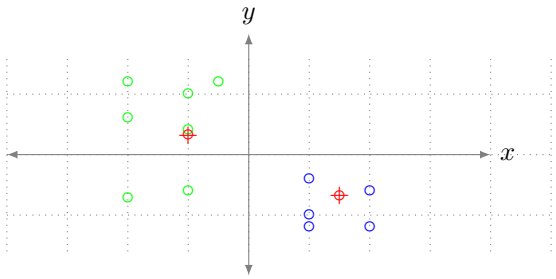


Figure 7: cluster centroid update in iteration 3

Under this configuration, subsequent cluster assignment do not change (hence cluster centroid update do not change too).

3.2.1 Python demo time

this is where I would like to show you a python demo on K-means clustering.

3.2.2 analysis of K-means

What is the one-line objective function for K-means? Given a dataset of $\mathbf{x}_1 \dots \mathbf{x}_N$, we want to find a partition $\mathcal{S} = \{S_l\}_{l=1}^K$, such that to minimize the sum of distance between each data in a partition $\mathcal{S}_l$ with its centroid μ_l:

$$\begin{aligned} \arg\min_{\mathcal{S}} \sum_{l=1}^{K} \sum_{\mathbf{x} \in S_l} \|\mathbf{x} - \boldsymbol{\mu}_l\|^2 &= \arg\min_{\mathcal{S}} \sum_{l=1}^{K} |S_l| \times \frac{1}{|S_l|} \sum_{\mathbf{x} \in S_l} \|\mathbf{x} - \boldsymbol{\mu}_l\|^2 \\ &= \arg\min_{\mathcal{S}} \sum_{l=1}^{K} |S_l| \text{Var}[S_l] \end{aligned} \tag{9}$$

where μ_l is the mean of points in S_l. This is equivalent to minimizing the pairwise squared deviations of points in the same cluster:

alternatively:

$$
\begin{aligned}
\arg\min_{\mathcal{S}} \sum_{l=1}^{K} \sum_{\mathbf{x}\in S_l} \|\mathbf{x}-\boldsymbol{\mu}_l\|^2 &= \arg\min_{\mathcal{S}} \sum_{l=1}^{K} \frac{1}{|S_l|} \times |S_l| \sum_{\mathbf{x}\in S_l} \|\mathbf{x}-\boldsymbol{\mu}_l\|^2 \\
&= \arg\min_{\mathcal{S}} \sum_{l=1}^{K} \frac{1}{|S_l|} \times \sum_{\mathbf{x},\boldsymbol{y}\in S_l} \|\mathbf{x}-\boldsymbol{y}\|^2 \\
&= \arg\min_{\mathcal{S}} \sum_{l=1}^{K} \frac{1}{|S_l|} \times \sum_{\mathbf{x}\neq\boldsymbol{y}\in S_l} \|\mathbf{x}-\boldsymbol{y}\|^2
\end{aligned} \tag{10}
$$

The equivalence can be deduced from identity:

$$
n\sum_{i=1}^{n}\|\mathbf{x}_i-\mu\|^2 = \sum_{\mathbf{x}\neq\mathbf{y}\in S_i}\|\mathbf{x}-\mathbf{y}\|^2 \tag{11}
$$

to make it simple we prove this using the scalar version. However, you should also aware that you can easily change it to vector form by:

$$
\begin{aligned}
(x_i-x_j)^2 \rightarrow &\|\mathbf{x}_i-\mathbf{x}_j\|_2^2 \\
&= \mathbf{x}_i^\top\mathbf{x}_i + 2\mathbf{x}_i^\top\mathbf{x}_j + \mathbf{x}_j^\top\mathbf{x}_j
\end{aligned} \tag{12}
$$

$$
\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i-x_j)^2 &= \sum_{i=1}^{n}\sum_{j=1}^{n}[(x_i-\bar{x})-(x_j-\bar{x})]^2 \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[(x_i-\bar{x})^2 - 2(x_i-\bar{x})(x_j-\bar{x}) + (x_j-\bar{x})^2] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}\underbrace{(x_i-\bar{x})^2}_{\text{no } j} - 2\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i-\bar{x})(x_j-\bar{x}) + \sum_{i=1}^{n}\sum_{j=1}^{n}\underbrace{(x_j-\bar{x})^2)}_{\text{no } i} \\
&= n\sum_{i=1}^{n}(x_i-\bar{x})^2 - 2\sum_{i=1}^{n}(x_i-\bar{x})\underbrace{\sum_{j=1}^{n}(x_j-\bar{x})}_{=0} + n\sum_{j=1}^{n}(x_j-\bar{x})^2 \\
&= n\sum_{i=1}^{n}(x_i-\bar{x})^2 + n\sum_{j=1}^{n}(x_j-\bar{x})^2 \\
&= 2n\sum_{i=1}^{n}(x_i-\bar{x})^2
\end{aligned} \tag{13}
$$

which means:

$$
\begin{aligned}
\sum_{i=1}^{n}(x_i-\bar{x})^2 &= \frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i-x_j)^2 \\
&= 2\times\Big(\frac{1}{2n}\sum_{i<j}^{n}(x_i-x_j)^2\Big) \qquad \text{upper triangle, so } \frac{1}{2} \text{ and diagonal } x_i-x_i=0 \\
&= \frac{1}{n}\sum_{i<j}^{n}(x_i-x_j)^2 \\
&= \frac{1}{n}\sum_{i\neq i}(x_i-x_j)^2
\end{aligned} \tag{14}
$$

3.3 Mixture Densities

one of the drawback of K-mean algorithm is that it assume Ecledian distance for measuring centroid and data points. Mixture densities are in the form of:

$$p(\mathbf{X}|\Theta) = \sum_{l=1}^{k} \alpha_l p(\mathbf{X}|\theta_l) \qquad \text{where } \sum_{l=1}^{k} \alpha_l = 1$$
$$\text{and } \Theta = \{\theta_1, \ldots, \theta_l\} \tag{15}$$

we are particularly interested in Gaussian Mixture Model (k-mixture) (GMM), for data $\mathbf{X} = \{x_1, \ldots x_n\}$, we have:

$$p(\mathbf{X}|\Theta) = \sum_{l=1}^{k} \alpha_l \mathcal{N}(\mathbf{X}|\mu_l, \Sigma_l) \qquad \text{where } \sum_{l=1}^{k} \alpha_l = 1$$
$$\text{and } \Theta = \{\alpha_1, \ldots \alpha_k, \mu_1, \ldots \mu_k, \Sigma_1, \ldots \Sigma_k\} \tag{16}$$

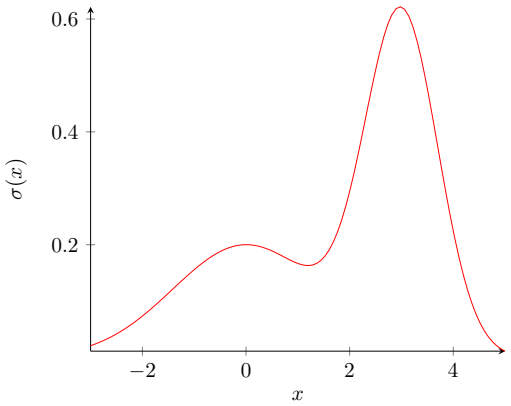


Figure 8: Example of a bi-modal Gaussian mixture model: $\mu_1 = 0, \mu_2 = 3.0, \sigma_1 = 2, \sigma_2 = 1, \alpha_1 = 0.4, \alpha_2 = 0.6$

3.3.1 Gaussian Mixture Model

They can be thought as the "softer" version of the K-means. Therefore, it has the exact number of steps:

1. **initialization**: initialize all the parameters $\alpha_1, \ldots \alpha_K, \mu_1, \ldots \mu_K, \Sigma_1, \ldots \Sigma_K$:

   then, in each iterations, one needs to perform:

2. **compute responsibility probability**:

$$p(l|x_i, \Theta^{(g)}) = \frac{\mathcal{N}(\mathbf{x}_i; \mu_l, \Sigma_l)}{\sum_{s=1}^{k} \mathcal{N}(\mathbf{x}_i|\mu_s, \Sigma_s)} \tag{17}$$

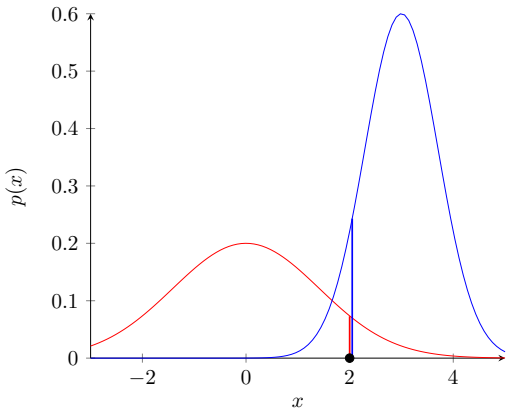


Figure 9: compute responsibility probabilities, red and blue each indicate its own unnormalized responsibilities

Looking at the figure (2), from this example, we can see that each vertical bar represents the unnormalized responsibility for each normal distribution. We need to get accountability through the sum of the two.

note that this corresponds to the cluster assignment step in K-means algorithm, except K-means has the "hard" assignment!

3. **update mixture density parameters** $\{\mu_l\}, \{\Sigma_l\}, \{\alpha_l\}$:

   We do so by update $\Theta^{(g)} \rightarrow \Theta^{(g+1)}$:

$$\alpha_l^{(g+1)} = \frac{1}{N} \sum_{i=1}^{N} p(l|\mathbf{x}_i, \Theta^{(g)})$$

$$\mu_l^{(g+1)} = \frac{\sum_{i=1}^{N} \mathbf{x}_i p(l|\mathbf{x}_i, \Theta^{(g)})}{\sum_{i=1}^{N} p(l|\mathbf{x}_i, \Theta^{(g)})}$$

$$\Sigma_l^{(g+1)} = \frac{\sum_{i=1}^{N} (\mathbf{x}_i - \mu_l^{(i+1)})(\mathbf{x}_i - \mu_l^{(i+1)})^\top p(l|\mathbf{x}_i, \Theta^{(g)})}{\sum_{i=1}^{N} p(l|\mathbf{x}_i, \Theta^{(g)})}$$

   or in the case of 1-D:

$$\sigma_l^{2\,(g+1)} = \frac{\sum_{i=1}^{N} (\mathbf{x}_i - \mu_l^{(i+1)})^2 p(l|\mathbf{x}_i, \Theta^{(g)})}{\sum_{i=1}^{N} p(l|\mathbf{x}_i, \Theta^{(g)})} \tag{18}$$

Can you think about how to make the Gaussian Mixture Model update **exact** the same as the K-means algorithm? The answer is that if $\Sigma = \mathbf{0}$, then K-means is exactly equivalent to a Gaussian mixture model. If you haven't figured it out yourself, I'll explain it to you in class.

3.3.2 convergence of the E-M algorithm

It is worth noting that the above update correspond to the update using E-M algorithm:

$$\Theta^{(g+1)} = \arg\max_{\Theta}\Big(\int_z \log\left(p(X,Z|\Theta)\right)p(Z|X,\Theta^{(g)})\mathrm{d}z\Big) \tag{19}$$

where:

$$\begin{aligned} p(X,Z|\Theta) &= \prod_{i=1}^{n} p(x_i,z_i|\Theta) = \prod_{i=1}^{n} \alpha_{z_i}\mathcal{N}(\mu_{z_i},\Sigma_{z_i}) \\ p(Z|X,\Theta) &= \prod_{i=1}^{n} p(z_i|x_i,\Theta) = \prod_{i=1}^{n} \frac{\alpha_{z_i}\mathcal{N}(\mu_{z_i},\Sigma_{z_i})}{\sum_{l=1}^{k}\alpha_l\mathcal{N}(\mu_l,\Sigma_l)} \end{aligned} \tag{20}$$

You can check out my online videos and GitHub lecture notes for its convergence proof

4 Hierarchical clustering

We will show an example through **Latent Dirichlet Allocation (LDA)**. In LDA the two most important assumption made are, given a document corpus, we assume:

1. each document has a latent distribution of topics
2. each topic has a latent distribution of words

4.1 topic modeling in general

If one was to have a paragraph:

Hong Kong Baptist University (HKBU) is a publicly funded tertiary liberal arts institution with a Christian education heritage. It was established as Hong Kong Baptist College with the support of American Baptists, who provided both operating and construction funds and personnel to the school in its early years. It became a public college in 1983.

if one needs to perform some data mining task to this. One must be able to convert the entire paragraph into a vector, i.e., an array of numbers. In my past industry project. I also need to compare paragraphs of job advertisements with paragraphs of course descriptions. Therefore, I must have an effective method to convert paragraphs to vector. In machine learning, the process of converting something into a vector is called "embedding".

4.2 Latent Dirichlet Allocation

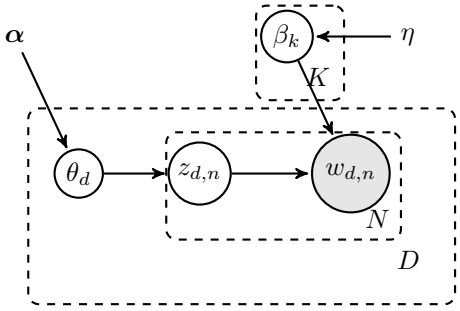


Figure 10: probabilistic graphical model for LDA

Still remember probabilistic graphical model from Lecture 4? Remember it shows you all the conditional independence? You also meant to define the probability of each of the variable condition on its parents. So what are they? Let me show you:

4.2.1 what is a plate?

a plate used in probabilistic graphical model is used so that you do not need to repeat the nodes:

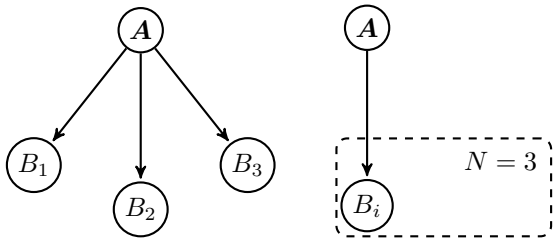


Figure 11: explain plate: the above two graphical models are identical

4.2.2 what are the observable?

Well, the only thing you can observe are words in a paragraphs. Look at figure (10), the node $w_{d,n}$ is observed. It means the n^{th} words in the d^{th} document. For example:

1. d_1:

   The Hong Kong Baptist College was opened on 11 September 1956 on the premises of the Pui Ching Middle School, a Baptist-affiliated secondary school, with an initial enrolment of 152 students. In 1957 the government gave the college land on Waterloo Road to build a permanent campus, which was completed in 1966.

2. d_2:

   The third reading of the Hong Kong Baptist College (Amendment) Bill was passed by the Legislative Council on 16 November 1994, transforming the college into the Hong Kong Baptist University.

   then $w_{1,5}$ = "College" and $w_{2,8}$ = "Baptist" (actually, this is not entirely correct, as we typically tokenize the words, for example, "andy lau" appears in a single token. However, for illustration purpose in this subject, let's assume each token is a word).

4.2.3 distribution of topics

Suppose there are K distinct topics in a document corpus (collection of documents). For example, "Entertainment", "Sports", etc. (remember toics are latent). Then each topic should have a different word distribution. For example, under the "Entertainment" topic, the token "andy lau" might have a higher probability and the token "Siobhán Haughey" would have a lower probability, whereas under the "Sports" topic, the token "'Siobhán Haughey" might have high probability and token 'andy lau" may have low probability.

Therefore, we let β_k to be a distribution of words for topic k, since β_k is a distribution, therefore, we need a distribution of distribution to model its prior density. A Dirichlet distribution does its job:

$$\beta_k \sim \text{Dir}(\eta, \dots \eta) \text{ for } k \in \{1, ..., K\} \tag{21}$$

where

$$\text{Dir}(\alpha_1, \dots, \alpha_K) = \frac{\Gamma\left(\sum_{i=1}^{K} \alpha_i\right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} x_i^{\alpha_i - 1} \tag{22}$$

4.2.4 modeling of words within document

Here is the target piece of LDA, we assume that each document has a latent distribution of topics.

So each document must have a K-dimensional probability vector associated with it. By the way, once we obtain that, we are done with vector embedding for that document.

For each document d:

associate it with a distribution of topics θ_d:

$$\theta_d \sim \text{Dir}(\alpha, \ldots, \alpha) \tag{23}$$

For each word $w_{d,n}$ in the document d:

generate the topic $z_{d,n}$ for the word $w_{d,n}$:

$$z_{d,n} \sim \text{Mult}(\theta_d) \tag{24}$$

generate the word $w_{d,n}$

$$w_{d,n} \sim \text{Mult}(\beta_{z_{d,n}}) \tag{25}$$

4.2.5 numerical example of LDA

let me illustrate using token Ids instead of words. We assume $K = 4$ and each document has $N = 20$ tokens, and the vocabulary of the corpus has 10 words/token $\{1, \ldots 10\}$

1. for $\{\beta_k\}$: we should have $\beta_1, \beta_2, \beta_3, \beta_4$, each is a random discrete probability vector of 10-dimensions. Remember it's the distribution of words for each topic. Let's choose hyper-parameter $\eta = 1$ and we obtain our samples:

$$\begin{aligned}
\beta_1 &\sim \text{Dir}(1,1,1,1,1,1,1,1,1,1) \\
&= [0.011 \quad 0.059 \quad 0.226 \quad 0.232 \quad 0.156 \quad 0.047 \quad 0.018 \quad 0.157 \quad 0.048 \quad 0.046] \\
\beta_2 &\sim \text{Dir}(1,1,1,1,1,1,1,1,1,1) \\
&= [0.152 \quad 0.149 \quad 0.02 \quad 0.065 \quad 0.08 \quad 0.131 \quad 0.008 \quad 0.18 \quad 0.166 \quad 0.049] \\
\beta_3 &\sim \text{Dir}(1,1,1,1,1,1,1,1,1,1) \\
&= [0.048 \quad 0.049 \quad 0.174 \quad 0.246 \quad 0.006 \quad 0.062 \quad 0.123 \quad 0.157 \quad 0.041 \quad 0.094] \\
\beta_4 &\sim \text{Dir}(1,1,1,1,1,1,1,1,1,1) \\
&= [0.129 \quad 0.017 \quad 0.059 \quad 0.031 \quad 0.02 \quad 0.223 \quad 0.148 \quad 0.127 \quad 0.114 \quad 0.13]
\end{aligned} \tag{26}$$

   By the way, if you know how to write python, then you can sample a Dirichlet random variable in the following way:

```
np.random.dirichlet([1,1,1,1,1,1,1,1,1,1],4)
```

   This is how I generate four random probability vectors in the Equation (26).

2. let's have 3 documents in the document corpus:

   (a) document $d = 1$:

$$\begin{aligned}
\text{tokens: } & [1,8,8,9,8,5,10,6,1,8,9,6,6,4,1,9,6,2,1,6] \\
\text{topic distributions: } \theta_1 &\sim \text{Dir}(2,2,2,2) = [0.199, 0.492, 0.207, 0.102]
\end{aligned} \tag{27}$$

(b) document $d = 2$,

$$\begin{aligned} \text{tokens: } & [1, 8, 3, 5, 1, 4, 1, 1, 9, 5, 1, 6, 9, 10, 1, 9, 9, 8, 1, 6] \\ \text{topic distributions: } \theta_2 & \sim \text{Dir}(2, 2, 2, 2) = [0.362, 0.392, 0.154, 0.091] \end{aligned} \tag{28}$$

(c) document $d = 3$,

$$\begin{aligned} \text{tokens: } & [10, 7, 3, 1, 3, 9, 1, 3, 8, 3, 6, 6, 3, 1, 3, 1, 1, 4, 8, 8] \\ \text{topic distributions: } \theta_3 & \sim \text{Dir}(2, 2, 2, 2) = [0.277, 0.195, 0.108, 0.42] \end{aligned} \tag{29}$$

By the way, I used the following python command to generate words. (But note that in real application, the words in the document are given, not randomly generated).

```
np.random.choice(10,20,p=np.random.dirichlet([1,1,1,1,1,1,1,1,1,
1]))+1
```

also for the distribution of topics of each document, I used the hyper-parameter $\alpha = 2$, I have generated the topic distribution $\{\theta_d\}$ using the following command:

```
np.random.dirichlet([2,2,2,2],3)
```

With all this information, now we can calculate the probability of each word/token, e.g.,:

1. $w_{2,4} = 5$ (document 2, word 4)

   (a) let's sample its corresponding topic:

$$\begin{aligned} z_{2,4} &\sim \text{Mult}(\theta_2) \\ &\sim \text{Mult}\big([0.362, 0.392, 0.154, 0.091]\big) \\ &= 2 \end{aligned} \tag{30}$$

   (b) so we can compute:

$$\begin{aligned} \Pr(w_{2,4}) &= \beta_{z_{2,4}}(5) \\ &= \beta_2(5) \quad \text{means the probability of token 5 under Topic 2, i.e., } \beta_2 \\ &= 0.08 \end{aligned} \tag{31}$$

2. $w_{3,6} = 9$ (document 3, word 6)

   (a) let's sample its corresponding topic:

$$\begin{aligned} z_{3,6} &\sim \text{Mult}(\theta_2) \\ &\sim \text{Mult}\big([0.277, 0.195, 0.108, 0.42]\big) \\ &= 1 \end{aligned} \tag{32}$$

   (b) so we can compute:

$$\begin{aligned} \Pr(w_{3,6}) &= \beta_{z_{3,6}}(9) \\ &= \beta_1(9) \quad \text{means the probability of token 9 under Topic 1} \\ &= 0.048 \end{aligned} \tag{33}$$

4.2.6 LDA in the real world

Of course, in real applications, all you have is which words/tokens are in each of the documents, you do not know what $\{\beta_k\}$, $\{\theta_d\}$ and $\{z_{d,n}\}$. Therefore, you need to make inference on:

$$\Pr\left(\{\beta_k\}, \{\theta_d\}, \{z_{d,n}\} | \{w_{d,n}\}\right) \tag{34}$$

This is usually achieved by using Monte Carlo Markov Chains (MCMC) or variational inference. I will be teaching both in an upcoming topic MATH4225.

5 Word Embedding

Remember, I talked about converting things into vectors called "embedding". Now that we understand how to embed documents, let's see how we should embed individual words. By the way, Natural Language Processing (NLP) is one of the most important/exciting applications of artificial intelligence, machine learning and data mining. Contrary to computer vision, NLP will influence the work of many people in the future.

5.1 word embedding introduction

words are symbols: one may **not** able to perform arithmetic operations on them. Using the terms we have studied. They are suppose to be norminal attributes. (remember topic 2?)

However, in many NLP tasks, we need to operate on them as if they are vectors. So we must turn each word into a **vector**, e.g., "machine" $\rightarrow$ [2.4, 1.2, 1.9 ...]

Once we do so, we can measure how similar or dissimilar between them. We can even perform "arithmetic" on them. A classic example from the the original word2vec paper would be:

$$\text{vec(King)} - \text{vec(man)} + \text{vec(woman)} = \text{vec(Queen)} \tag{35}$$

Mikolov et. al., (2013) "Linguistic Regularities in Continuous Space Word Representations"

5.1.1 first attempt: one-hot encoding/embedding

a very naive and simple approach is to just use one-hot embedding, after all, students should be very familiar with one-hot vector by now:

$$\begin{bmatrix} \text{“}a\text{”} & 1 & 0 & \cdots & 0 \\ \text{“}abbreviate\text{”} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \text{“}zoology\text{”} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

however, the structure is huge and sparse. Think about the length of those vectors, they are as long as the total number of words in the vocabulary. In English, there is estimated of $40,000$ words.

In addition to the storage waste, one other disadvantage is that every pair of words are exactly $\sqrt{2}$ apart. So you can't really measure the similarity or dissimilarity between them.

The question is, can we do better?

5.2 second attempt: word2vec algorithm

The only data required are paragraphs of words, i.e., no human labeling is required. Unlike LDA models, with word-embeddings, we don't even need to know which word appear in which document.

5.2.1 conditional density using (target $\leftrightarrow$ context)

Although the paragraphs of words are the only information for the word2vec algorithm, however, we have the position of each word in the document. Therefore, we will use this information alone to construct the learning task:

1. Word2Vec algorithm leverage (*"target", "context"*) relationships to maximize the conditional probabilities.
2. which way should the condition be? the answer is both. Therefore it offers two approaches, i.e., to maximize one of the two conditional densities:
   (a) skip-gram: $\Pr(\text{"context"}|\text{"target"})$
   (b) continuous bag of words (CBOW): $\Pr(\text{"target"}|\text{"context"})$

So we need a methodology to construct context and target:

1. pick window size (odd number)
2. extract all tokens based on this chosen window size
3. remove middle word in each window; this becomes your **target** word, rest are **context**

An important question to consider is how to define this probability? Taking Skip-gram as an example (for CBOW, you can calculate it similarly), it is $\Pr(\text{"context"} = \text{"a specific word"}|\text{"target"})$, so this discrete probability should be defined on every word in the vocabulary. So it should return a probability vector as large as the vocabulary size!

What function should we consider to construct such a probability vector? you guessed right! Since we need a discrete probability, we can use the Softmax function. Let us generically define these probabilities in terms of the "center" (c) and the "output" word (o):

$$\frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \tag{36}$$

and the entire softmax vectors can be represented as:

$$\left(\frac{\exp(\mathbf{u}_1^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}, \cdots, \frac{\exp(\mathbf{u}_{o^\star}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}, \cdots, \frac{\exp(\mathbf{u}_{|\mathcal{V}|}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \right) \tag{37}$$

and the index $o^\star$ is the one we are interested in.

5.2.2 skip-gram example: building training set

let's try **Skip-gram** of a window size 3, from the following sentence:

"the cat sit on the mat"

1. firstly, we generate all possible windows:
   (a) "the", "cat", "sit", target: cat
   (b) "cat", "sit", "on" , target: sit
   (c) "sit", "on ", "the", target: on
   (d) "on" , "the", "mat", target: the

   as far as this note is concerned, we assume that for each target word, there are all context words on both sides. Therefore the word "the" will not be a context word

2. from each of the windows, the algorithm generate the input and output pairs for maximizing the conditional probabilities

$$(\mathbf{x}, y) = \text{(target, context)} \tag{38}$$

therefore we obtained:

$$\begin{array}{rcl} \text{“the”} & \leftarrow & \text{“cat”} \\ \text{“sit”} & \leftarrow & \text{“cat”} \\ \text{“cat”} & \leftarrow & \text{“sit”} \\ \text{“on”} & \leftarrow & \text{“sit”} \\ \text{“sit”} & \leftarrow & \text{“on”} \\ \text{“the”} & \leftarrow & \text{“on”} \\ \text{“on”} & \leftarrow & \text{“the”} \\ \text{“mat”} & \leftarrow & \text{“the”} \end{array} \tag{39}$$

3. Using some initialization values, it is possible to take each context, target pair and maximize their conditional probability. Note that we need to maximize a number of conditions, each word has a chance to be a context and target word:

$$\begin{aligned} &\Pr\left(\text{“the”}|\text{“cat”}\right) \times \Pr\left(\text{“sit”}|\text{“cat”}\right) \times \Pr\left(\text{“cat”}|\text{“sit”}\right) \times \Pr\left(\text{“on”}|\text{“sit”}\right) \times \\ &\Pr\left(\text{“sit”}|\text{“on”}\right) \times \Pr\left(\text{“the”}|\text{“on”}\right) \times \Pr\left(\text{“on”}|\text{“the”}\right) \times \Pr\left(\text{“mat”}|\text{“the”}\right) \end{aligned} \tag{40}$$

we can express the above mathematically as, when choosing a window size $2m+1$:

$$\begin{aligned} \mathcal{L} &\equiv \prod_{n=1}^{N} \prod_{-m \le j \le m, j \neq 0} \Pr\left(w_{n+j}|w_n\right) \\ \log(\mathcal{L}) &= \sum_{n=1}^{N} \sum_{-m \le j \le m, j \neq 0} \log\Big(\Pr\left(w_{n+j}|w_n\right)\Big) \end{aligned} \tag{41}$$

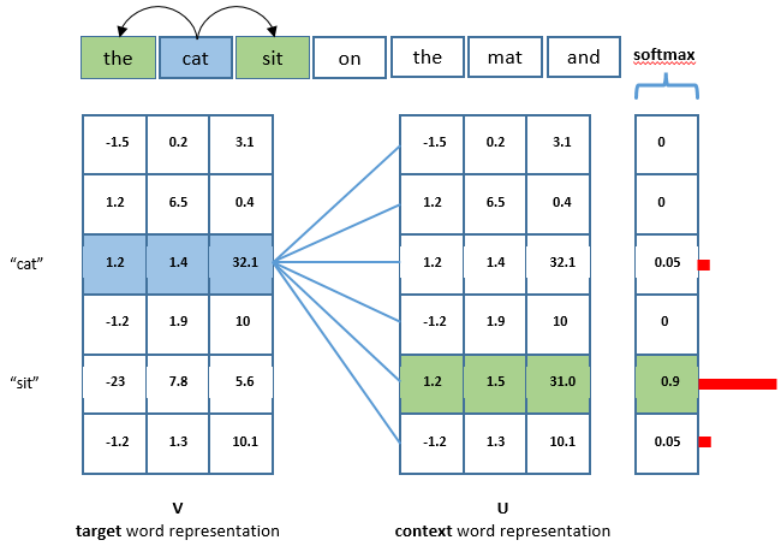
the outer sum tells us that each word n will get to be the “target” word once. The inner product is all the conditional densities using that target word.

4. For each word w_n, it has 2 representations $\mathbf{u}_n$ and $\mathbf{v}_n$, one for output (o) and one for context (c), and of course their size is much smaller than using one-hot!

For example, looking at a particular (i.e, the 2[nd] term in the product in Eq.(40):

$$\Pr\left(o = \text{“sit”} \,|\, c = \text{“cat”}\right) \tag{42}$$

we need to maximize the conditional density of $o =$ context word given a $c =$ target word. We illustrate the above using the diagram below:



5.2.3 optimizing Skip-Gram objective function

we need to maximize the probability of $c =$ context word given a $t =$ target word.

In order to compute:

$$\arg\max_{\{\mathbf{v}\},\{\mathbf{u}\}} \Big\{ \mathcal{L} \equiv \sum_{n=1}^{N} \sum_{-m \le j \le m, j \neq 0} \log\Big(\Pr\left(w_{n+j}|w_n\right)\Big) \Big\} \tag{43}$$

We need to calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_t}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{u}_c}$, $\forall \mathbf{v}_t, \mathbf{u}_c \in \mathcal{V}$. However, due to symmetry, we only look at $\arg\max_{\mathbf{v}_t}$. Also, let's look at the derivative of a single term in the sum, where we usually write:

$$\Pr\left(w_{n+j}|w_n\right) \equiv \Pr(o|c) \tag{44}$$

$$
\begin{aligned}
\log(\Pr(o|c)) &= \log\Big(\frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\Big) \\
\frac{\partial \log(\Pr(o|c))}{\partial \mathbf{v}_c} &= \frac{\partial \mathbf{u}_o^\top \mathbf{v}_c}{\partial \mathbf{v}_c} - \frac{\partial \log\Big(\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)}{\partial \mathbf{v}_c} \\
&= \mathbf{u}_o - \Big(\frac{\partial}{\partial \mathbf{v}_c}\log\Big(\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\Big) \\
&= \mathbf{u}_o - \Big(\frac{1}{\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\frac{\partial}{\partial \mathbf{v}_c}\Big(\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\Big) \\
&= \mathbf{u}_o - \Big(\frac{1}{\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\Big(\sum_{o'\in\mathcal{V}}\frac{\partial}{\partial \mathbf{v}_c}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\Big) \\
&= \mathbf{u}_o - \frac{1}{\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\Big(\sum_{o'\in\mathcal{V}}\mathbf{u}_{o'}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big) \\
&= \mathbf{u}_o - \frac{\sum_{o'\in\mathcal{V}}\mathbf{u}_{o'}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}{\sum_{o'\in\mathcal{V}}\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \\
&= \mathbf{u}_o - \sum_{o'\in\mathcal{V}}\frac{\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}{\sum_{o''\in\mathcal{V}}\exp(\mathbf{u}_{o''}^\top \mathbf{v}_c)}\mathbf{u}_{o'} \\
&= \mathbf{u}_o - \sum_{o'\in\mathcal{V}}\Pr(o'|c)\mathbf{u}_{o'} \\
&= \mathbf{u}_o - \mathbb{E}_{o'\sim\Pr(o'|c)}[\mathbf{u}_{o'}]
\end{aligned}
\tag{45}
$$

Obviously, when $|\mathcal{V}|$ is too large, the computational cost of computing the sum can be too high, there are many NLP mechanisms that can help us reduce the amount of computation. We will discuss them in other topics!

5.3 Simple CBoW example

very similar to to predict target word given multiple context words, where $\mathbf{u}_c$ become the average of context vectors.