Università di Genova | **DIBRIS** DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Learning relational models from networked data on the web

Davide Riva

Master Thesis

Università di Genova, DIBRIS Via Opera Pia, 13 16145 Genova, Italy
https://www.dibris.unige.it/

Università di Genova

MSc Computer Science

Data Science and Engineering Curriculum

Learning relational models from networked data on the web

Davide Riva

Advisors: Annalisa Barla, Davide Garbarino Examiner: Nicoletta Noceti

August, 2020

Abstract

Network inference is the process of uncovering relations between data, particularly effective in biology to reconstruct Gene Regulatory Networks (GRN). Supervised methods were also proposed in the past to model a network of documents and predict links between them.

In this thesis, we propose an unsupervised approach to extract an undirected and unweighted graph given a collection of texts. In particular, we present a pipeline to construct and use a dataset composed of pages of a given website.

I wish to express my deepest gratitude to all students and professors who went along with me on this journey. Among those, a special thanks to Eros Viola, Riccardo Bianchini, Olga Matthiopoulou, Marco Rando, Andrea Canepa, Larbi Touijer, Federico Lodovici, Roberto Castellotti, Antonio Vivace, Marco Orsingher, Michele Pugno, Adrian Castro, Annalisa Barla, Nicoleta Noceti and Davide Garbarino.

In addition, I would like to thank Jordan Boyd-Graber and David M. Blei who released for free their lessons, talks and material which were useful during the writing of this thesis.

I am also indebted to Fabio Antonio Stella, which made me fall in love with this subject and played an important role in the choices I made that led me here.

I wish to acknowledge the support and great love of Laura and her family, which helped me tackle this degree course which is going to end.

Last but not least, I would like to recognize the invaluable economic assistance that my family provided during my study.

Thank you all.

Contents

Chapter 1

Introduction and motivation

In the field of biology, unsupervised methods like ARACNE (see [MNB^{+}06]) are used to obtain Gene Regulatory Networks starting from microarray expression profiles.

Methods like Relational Topic Model (introduced in [CB09]) were also proposed to model a collection of documents and the links between them in a supervised way.

The goal of this thesis is to propose a novel approach to perform network inference given a collection of webpages scraped from a website, exploiting ideas from the two methods mentioned before. In particular, the final result is an unweighted and undirected graph which connects pages with similar content.

The manuscript is organized as follows: in Chapter 2 we present some techniques to encode words numerically. In Chapter 3 we move from word to document representations and we describe Topic Modeling. In Chapter 4 we describe network inference, presenting ARACNE. Chapter 5 and Chapter 6 describe our original approach, outling some possible future directions. Finally, Appendices A, B, C and D cover a part of the background knowledge required to read this document.

Chapter 2

Word Embedding

Documents are made by words. To be manipulable by an algorithm these words need to be represented in a numerical format.

This chapter covers some techniques able to fulfil this task, pointing out the advantages and disadvantages of choosing one instead of another.

To avoid misunderstandings we define the term "word" as a word in a particular position in a document, while we use the word "term" as a generic word into a dictionary.

For instance, given a document "A cat is an animal but an animal is not a cat" its list of words is $w = (a, cat, is, an, animal, but, an, animal, is, not, a, cat)$ while its set of terms is $d = \{a, cat, is, an, animal, but, not\}$.

2.1 One Hot Encoding

Given a list of words $w = (w_1, w_2, \dots, w_N)$, we first generate a dictionary of unique terms $d = \{d_1, d_2, ..., d_K\}$ with an arbitrary order between words. Note that $K \leq N$ is always true and $K = N$ happens only when there is no repetition of words in the documents.

Each term d_i in the dictionary can now be encoded as $x_i \in \{0, 1\}^K$ following the One Hot Encoding approach. All elements of x_i are zero except a "1" in the position i of the vector. That means that two words w_l, w_j in different positions will have the same value x_i if and only if they correspond to the same term d_i.

Applying One Hot Encoding on the previous example presented in the introduction of this chapter leads to the following values of x_i:

- $x_1 = x_a = [1, 0, 0, 0, 0, 0, 0]$
- $x_2 = x_{cat} = [0, 1, 0, 0, 0, 0, 0]$
- $x_3 = x_{is} = [0, 0, 1, 0, 0, 0, 0]$
- $x_4 = x_{an} = [0, 0, 0, 1, 0, 0, 0]$
- $x_5 = x_{animal} = [0, 0, 0, 0, 1, 0, 0]$
- $x_6 = x_{but} = [0, 0, 0, 0, 0, 1, 0]$
- $x_7 = x_{not} = [0, 0, 0, 0, 0, 0, 1]$

Despite its simplicity and the fact that it is not computationally expensive to use, it has some disadvantages:

- vectors are sparse: only one element of each vector is not zero
- there is no concept of similarity: all vectors have the same distance between each other

2.2 Word2vec

Word2vec is a family of approaches to overcome the disadvantages of One Hot Encoding using neural networks. Originally presented in [MCCD13], it was later resumed by other authors due to the lack of details in the original paper. For our document, we use [Ron14] as a reference.

In this section, two architectures are proposed. Both of them should be trained using mini-batch gradient descent instead of gradient descent to allow the algorithm to scale when datasets grow in size.

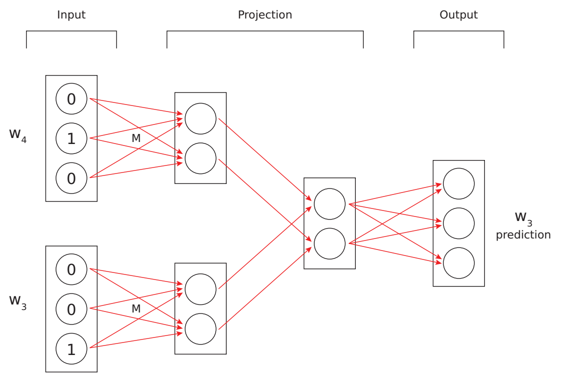
After a learning phase, they will be able to map each term $x_i \in \{0, 1\}^K$ to a point $y_i \in \mathbb{R}^V$, with $V \ll K$. Points in the new space will have a notion of similarity: if two terms d_l and d_j appear in similar contexts and/or they have the same semantic content, their vectors y_l and y_j will be close together. On the opposite situation, y_l and y_j will be distant from each other.

For instance, y_{cat} will probably be closer to y_{dog} than $y_{computer}$.

2.2.1 Continuous Bag-of-Words Model (CBOW)

First, the entire corpus is divided into contiguous blocks of words with odd length N called n-grams.

Suppose to have the phrase "A cat is the best animal." and n-grams of size 3. The corresponding blocks are:



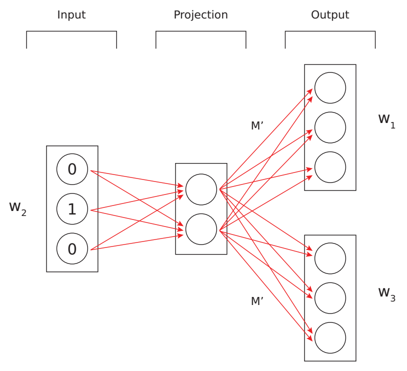


Figure 2.1: CBOW (left) and Skip-Gram (right) models with $K = 3$ and $V = 2$.

- (a, cat, is)
- (cat, is, the)
- $(is, the, best)$
- $(the, best, animal)$

The goal is to predict a word given its context. For each n-gram, the CBOW neural network has as inputs all words in the block except the one in the center which is used as ground truth for the classification task. All words are encoded using the One Hot Encoding method.

For each n-gram, the first layer takes each input word encoded as $x_i \in \{0, 1\}^K$ and projects it into a smaller space $\mathbb{R}^V$ where V is a hyperparameter to tune. This is done through a shared matrix $M_{K \times V}$ which represents a fully connected layer. Each projection is then averaged out with the others in the same block. Note that in this way the order of the input words of the same sequence does not matter anymore.

The previous layer is then linked to a final layer in a fully connected way. Since the classification task is on a One Hot Encoding vector, a softmax activation function is used at the end.

Instead of using the network for the task for which it was trained, we use the matrix $M_{K \times V}$ to encode all terms as vectors in $\mathbb{R}^V$.

Note that there is no need to compute Mx_i for each i. Since every x_i has zero values everywhere except a "1" in the position i of the vector, the result of the multiplication Mx_i is the i-th row i the matrix M. For this reason, each row i of M represents the term in position i of the dictionary, encoded using CBOW.

An illustration of CBOW can be seen in Figure 2.1.

2.2.2 Continuous Skip-gram Model (Skip-gram)

The Skip-gram model is the opposite approach of CBOW.

For each n-gram, the word in the center is fed to the neural network as the only input.

The second layer projects it to a smaller space through a matrix $M_{K \times V}$. Unlike the previous model, there is no averaging since there is just one word as input.

Finally, the last layer tries to predict all remaining words of a block. Each predicted word is obtained multiplying the hidden layer with a shared matrix $M'_{V \times K}$. For each group of K nodes which refers to a predicted word, we apply separately a softmax function. To increase the accuracy, we can pick the number of output words less than $N-1$ and then sampling words in the same phrase weighted by how distant they are from the input word.

If there are $N-1$ words to predict and each word is represented as a vector in $\mathbb{R}^K$, the total number of nodes of the final layer is $(N-1)K$.

As previously, we use the matrix M to encode each term x_i.

An illustration of Skip-Gram can be seen in Figure 2.1.

2.2.3 Interpretation of the results

CBOW and Skip-Gram models are similar to Autoencoders, with the difference that we are using the intermediate representations to predict words in the same phrase. This leads to words having close values of y if they appear often in the same context.

Furthermore, the paper [MCCD13] found out that these models go beyond syntactic regularities: for example, the result of $y_{king} - y_{man} + y_{woman}$ is the closest vector to y_{queen}. Note that subsequent papers like [NvNvdG19] scale down the previous claim, adding the constraint that no input vector can be returned by the prediction.

2.3 Global Vectors (GloVe)

GloVe was proposed in [PSM14] to overcome the limitations of Word2vec. In particular, it builds a term-term co-occurrence matrix to utilize the statistics of the entire corpus instead of training the model on single n-grams.

Let X the term-term co-occurrence matrix. X_{ij} measures how many times the term d_i occurs in the corpus near the term d_j. Based on how you define the term "near", the elements of the matrix will have different values. For simplicity, we increment the counter ($X_{ij} = X_{ij} + 1$) every time two terms appear in a phrase with less than N words between them, with N a hyperparameter to tune. Given that, X is symmetric.

An initial proposal could be to solve $\min_{u,v} \sum_{i,j} (u_i^T v_j - X_{ij})^2$ in order to have vectors $u_i, v_j \in \mathbb{R}^V$ that have a high dot product if the terms d_i and d_j appear frequently together and vice versa.

We then apply the logarithm to X_{ij}:

$$\min_{u,v} \sum_{i,j} (u_i^T v_j - log_2(X_{ij}))^2$$

In this way, we lower the gap between frequent terms and rare ones.

The huge problem with this model is that all terms are weighted in the same way, no matter how frequent they appear in the corpus. To avoid this drawback, we introduce a weighting function $f(X_{ij})$ with the following characteristics:

- $\boldsymbol{f(0) = 0}$: when $X_{ij} = 0$ the previous equation is undefined, while here the co-occurrence is not considered
- $\boldsymbol{f}$ **should be non-decreasing**: frequent terms must be considered more than rare ones
- $\boldsymbol{f}$ **needs to have a cutoff** to avoid weighting too much terms that appear often in the text and which do not convey semantic information. These terms are commonly called "stop words" in the literature. Examples are "the", "a", "from" and so on.

An illustration of a valid $f(X_{ij})$ can be viewed in Picture 2.2.

After this addition, the minimization problem becomes:

$$\min_{u,v} \sum_{i,j} f(X_{ij})[u_i^T v_j - \log_2(X_{ij})]^2$$

Consider that even the summation is over all combinations of terms, the matrix is sparse and so the minimization is not as computationally intensive as computing it on a dense matrix.

After taking gradient descent to resolve that minimization, we will obtain vectors u_j, v_j similar between each other because of the symmetricity of X. Since $u_j \simeq v_j$, a reasonable choice for our new vector representation of each term d_j could be taking their average:

$$y_j = \frac{u_j + v_j}{2}$$

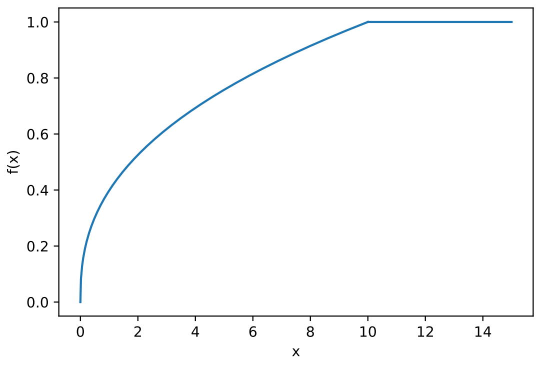


Figure 2.2: Example of a valid $f(X_{ij})$ function for the minimization step of GloVe.

2.4 Measuring distances between terms

In the previous sections the term "close" was intentionally left vague. Different distances between vectors could be considered. In particular, we propose two of them:

- **euclidian distance**: $distance(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$
- **cosine similarity**: $distance(p, q) = \frac{p^T q}{\|p\|\|q\|} = \frac{\sum_i p_i q_i}{\sqrt{\sum_i p_i^2}\sqrt{\sum_i q_i^2}}$

While the first one measures the straight-line distance of two points, the second one returns the cosine of the angle between the corresponding vectors.

In the literature presented, the cosine similarity is the default choice to measure the closeness between terms. One reason to prefer it instead of the euclidian distance can be found in [SW15]: the magnitude of the vector representation of a term is correlated with the frequency of its appearance in the corpus which means that a term with the same semantical meaning of another could have a different magnitude if it appears more often in text than the other one.

Chapter 3

Topic Modeling

In Chapter 2, we presented some techniques to encode words as vectors. Starting from a Bag-of-Words model in which we assume that the order of the words inside a text does not matter, it is possible to follow a similar approach also for documents. Topic models play an important role in describing, organizing and comparing unstructured collections of documents.

The first step is to generate a matrix of term counts X for the entire corpus: each row is a document represented in the BOW format and each column represents the count of occurrence of a particular term in the documents.

The goal of the second step is to produce two matrices:

- **term-topic matrix**: each row represents how related each term is to all topics
- **document-topic matrix**: each row represents how related each document is to all topics

For instance, a document about bears is likely to have high intensity on some topics which in turn have high intensity on words related to animals; see Figure 3.2 for an example.

3.1 Latent Semantic Analysis (LSA)

Given the number of topics V as an hyperparameter, Truncated SVD is applied to the matrix obtained in the first step (see Figure 3.1 for a graphical representation):

$$X_{N\times K} = U_{N\times V}\Sigma_{V\times V}D^T_{K\times V}$$

The matrix U is then used as the document-topic matrix, while D represents the term-topic matrix.

Using U instead of X for comparing documents when $V \ll K$ leads to a smaller memory footprint. Furthermore, the terms are no longer orthogonal: thanks to this it is possible to compute the similarity between a term and a document even if the document does not contain the term itself. The key point to observe is that both terms and documents are described as topics: if a term is frequent in some topics and a document is frequent in the same topics, they will have a similar representation in the new space.

An overview of this technique can be found at [Dum04].

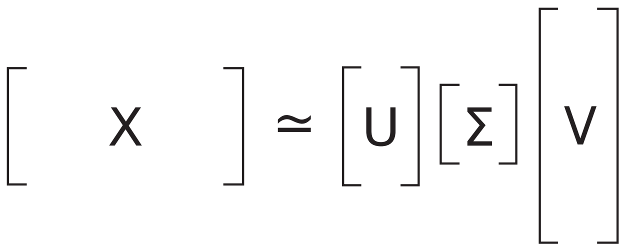


Figure 3.1: Truncated SVD represented graphically; matrices size proportions are kept intact

$$\text{Words}\left\{\begin{array}{r} \\ \text{Bear} \\ \text{Cat} \\ \text{Plane}\end{array}\right. \overset{\text{Topics}}{\overbrace{\begin{array}{cc} 1 & 2 \end{array}}} \quad \begin{bmatrix} 0.99 & 0.01 \\ 0.80 & 0.20 \\ 0.05 & 0.95 \end{bmatrix} \qquad \text{Topics}\left\{\begin{array}{r} 1 \\ 2 \end{array}\right. \overset{\text{Words}}{\overbrace{\begin{array}{cccc} \text{Animal} & \text{Furry} & \text{Technology} & \text{Space} \end{array}}} \begin{bmatrix} 0.60 & 0.35 & 0.05 & 0.00 \\ 0.01 & 0.01 & 0.50 & 0.48 \end{bmatrix}$$

Figure 3.2: Example of a term-topic and a document-topic matrix.

3.2 Latent Dirichlet Allocation (LDA)

LDA makes the same assumptions about topics as LSA but unlike the latter, it views topics and documents in a probabilistic way. In particular, a document is a categorical distribution over topics θ_d and a topic is a categorical distribution over words β_i. The probability of sampling a document is instead modelled as a Dirichlet distribution $Dir(\alpha)$.

A major drawback of LDA is that the number of topics is assumed to be known a-priori and treated as a hyperparameter.

Variants of this model are present in the literature: in this chapter, we use [Ble11] and [BNJ03] as a reference.

3.2.1 Generative process

In this section we describe LDA by its generative process, assuming we have already obtained the parameters of the model.

We define:

- $\boldsymbol{\beta_{1:K}}$: K topics, where each topic β_i is a distribution over the terms in the vocabulary
- $\boldsymbol{\theta_d}$: a document d described as a categorical distribution over topics, where $\theta_{d,k}$ is the topic proportion for topic k in document d
- $\boldsymbol{w_d}$: observed words of document d, where $w_{d,n}$ is the word in position n of document d

The generation of M documents with N words each can be summarized as:

1. $\theta_d \sim Dir(\alpha)$: pick a document
2. $z_{d,n} \sim Mult(\theta_d)$: sample a topic
3. $w_{d,n} \sim \beta_{z_{d,n}}$: pick a term
4. use that term as a word for the document you want to generate
5. loop N times to step 2
6. loop M times to step 1

LDA is a probabilistic graphical model and it is represented graphically in Figure 3.3.

The joint distribution is:

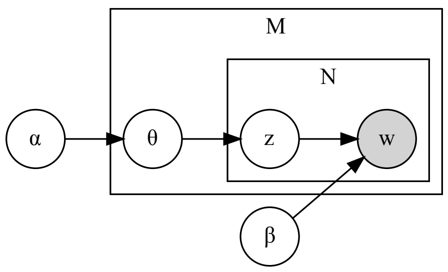
$$\begin{aligned} p(\theta, z, w|\alpha, \beta) &= \prod_d p_d(\theta, z, w|\alpha, \beta) \\ &= \prod_d [p(\theta_d|\alpha) \prod_n p(z_{d,n}|\theta_d) p(w_{d,n}|\beta, z_{d,n})] \end{aligned}$$

Given the distributions explained before, we can state that:

$$p(\theta_d|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_{d,k}^{\alpha_k - 1}$$

$$p(z_{d,n}|\theta_d) = \theta_{d,z_{d,n}}$$

$$p(w_{d,n}|z_{d,n}, \beta) = \beta_{z_{d,n},w_{d,n}}$$



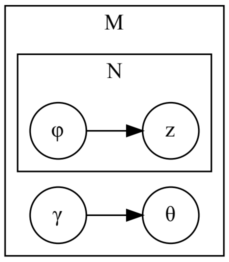


Figure 3.3: Graphical model representation of LDA (left) and graphical model representation of the variational distribution (right). Nodes are random variables, shaded if observed. An edge denotes a dependence between two random variables. Plates denote multiplicity.

3.2.2 Model inference

The goal of LDA is to automatically discover topics over a collection of documents. Since only words are observed, topics are considered latent variables.

Note that inference in this case can be seen as the reverse of the generative process explained in Section 3.2.1.

Mathematically speaking, inference becomes a question of computing the posterior distribution:

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)}$$

The denominator $p(w_{1:D}|\alpha, \beta)$ is the probability of observing the corpus under all possible topic model combinations and thus intractable to compute. For this reason, in this section we approximate the posterior distribution through a variational method (see Appendix D). In particular, we set q as:

$$\begin{aligned} q(\theta, z|\gamma, \phi) &= \prod_d q_d(\theta, z|\gamma, \phi) \\ &= \prod_d q_d(\theta|\gamma) q_d(z|\phi) \\ &= \prod_d [q(\theta_d|\gamma_d) \prod_n q(z_{d,n}|\phi_{d,n})] \end{aligned}$$

$q(\theta_d|\gamma_d)$ is governed by a Dirichlet distribution, while $q(z_{d,n}|\phi_{d,n})$ by a Categorical distribution. Its graphical representation can be seen in Figure 3.3.

To do model inference from a collection of documents, we start defining the ELBO function:

$$
\begin{aligned}
\ln[\prod_d p_d(w|\alpha,\beta)] &\geq \mathop{\mathbb{E}}_q\{\ln[p(w,\theta,z|\alpha,\beta)] - \ln[q(\theta,z|\gamma,\phi)]\} \\
&= \mathop{\mathbb{E}}_q\{\ln[\prod_d p_d(w,\theta,z|\alpha,\beta)] - \ln[\prod_d q_q(\theta,z|\gamma,\phi)]\} \\
&= \mathop{\mathbb{E}}_q\{\sum_d \ln[p_d(w,\theta,z|\alpha,\beta)] - \sum_d \ln[q_q(\theta,z|\gamma,\phi)]\} \\
&= \sum_d \mathop{\mathbb{E}}_q\{\ln[p_d(w,\theta,z|\alpha,\beta)] - \ln[q_q(\theta,z|\gamma,\phi)]\} \\
&= \sum_d \mathop{\mathbb{E}}_{q_d}\{\ln[p_d(w,\theta,z|\alpha,\beta)] - \ln[q_d(\theta,z|\gamma,\phi)]\}
\end{aligned}
$$

First, we fix α and β to find the best $q(\theta, z|\gamma, \phi)$ that minimizes the KL-divergence between q and p. Then, we fix $(\theta, z|\gamma, \phi)$ to find the best α and β that maximize the lower bound. This process, if done iteratively, converges to a local optimum. In the literature, it is called expectation-maximization (EM).

Since constrained maximization and the Newton-Raphson algorithm are needed to solve the problem instead of traditional variational inference, we intentionally present only a hasty description of the update algorithm:

1. initialize β randomly, set each $\phi_{d,n}$ as $\frac{1}{K}$ and each γ_d as $\alpha_d + \frac{N}{K}$
2. for each document, update the local latent variables γ and ϕ until convergence
3. update the global variables β and α for the entire corpus until convergence
4. loop to 2 until convergence

The updates are affected by:

- $\phi_{d,n,k} \propto \beta_{k,w_n} e^{\Psi(\gamma_{d,k}) - \sum_j^K \Psi(\gamma_{d,j})}$, where Ψ is the first derivative of the logarithm of the gamma function
- $\gamma_{d,k} = \alpha_k + \sum_n \phi_{d,n,k}$
- $\beta_{k,j} \propto \sum_d \sum_n \mathbb{1}_{[w_{d,n}==j]} \phi_{d,n,k}$
- $\frac{\partial ELBO}{\partial \alpha_k} = \sum_d [\Psi(\sum_j^K \alpha_j) - \Psi(\alpha_k) + \Psi(\gamma_{d,k}) - \Psi(\sum_j^K \gamma_{d,j})]$

The interested reader can find more details looking at Appendix A of [BNJ03].

Note that step 2 can be parallelized: each document has its local variables that are conditionally independent of the other local variables and data in different documents.

After the execution of the algorithm, each row d of the final document-topic matrix can be obtained using the Maximum Likelihood Estimation method over $q(\theta_d|\gamma_d)$.

3.2.3 Sparsity considerations

Using the Dirichlet governed by the α parameter to induce sparsity allows us to have documents described by fewer topics than using an uninformative prior. The same reasoning can be applied to topics, introducing a Dirichlet that governs the probability of having a topic with a particular set of terms.

Inducing sparsity combined with the number of topics much less than the number of documents lower the risk of obtaining two different topics with a similar distribution of words.

3.3 Online LDA

A major drawback of LDA is the impossibility to scale as the data grows: each iteration of the algorithm requires an entire scan of the corpus, which is computationally expensive.

For this reason, [HBB10] proposes an online learning variation based on the original LDA model. This algorithm called online LDA does not need to store all documents together, but they can arrive in a stream and be discarded after one look.

The main difference between the graphical model presented in Section 3.2.2 and this one is that the latter assumes each β_k sampled from a Dirichlet distribution as presented in Section 3.2.3:

$$\beta_k \sim Dir(\eta) \tag{3.1}$$

For this reason, q becomes:

$$\begin{aligned} q(\beta, \theta, z|\lambda, \gamma, \phi) &= [\prod_k q_k(\beta|\lambda)][\prod_d q_d(\theta, z|\gamma, \phi)] \\ &= [\prod_k q(\beta_k|\lambda_k)][\prod_d q_d(\theta|\gamma) q_d(z|\phi)] \\ &= [\prod_k q(\beta_k|\lambda_k)]\{\prod_d [q(\theta_d|\gamma_d) \prod_n q(z_{d,n}|\phi_{d,n})]\} \end{aligned}$$

$q(\beta_k|\lambda_k)$ and $q(\theta_d|\gamma_d)$ are governed by a Dirichlet distribution, while $q(z_{d,n}|\phi_{d,n})$ by a Categorical distribution.

Due to that change compared to LDA, the ELBO is now:

$$\ln[p(w|\alpha, \eta)] \geq \mathop{\mathbb{E}}_q\{\ln[p(w, z, \theta, \beta|\alpha, \eta)] - \ln[q(z, \theta, \beta|\gamma, \phi, \lambda)]\}$$

If we have M documents, the ELBO can be factorized as:

$$\{\sum_{d=1}^{M} \mathop{\mathbb{E}}_{q}[\ln p_d(w|\theta, z, \beta) + \ln p_d(z|\theta) - \ln q_d(z|\phi) + \ln p_d(\theta|\alpha) - \ln q_d(\theta|\gamma)]\}$$
$$+ \mathop{\mathbb{E}}_{q}[\ln p(\beta|\eta) - \ln q(\beta|\lambda)] =$$
$$\{\sum_{d=1}^{M} \mathop{\mathbb{E}}_{q}[\ln p_d(w|\theta, z, \beta) + \ln p_d(z|\theta) - \ln q_d(z|\phi) + \ln p_d(\theta|\alpha) - \ln q_d(\theta|\gamma)]$$
$$+ [\ln p(\beta|\eta) - \ln q(\beta|\lambda)]/M\}$$

Similar to what we do for LDA, we could maximize the lower bound using coordinate ascent over the variational parameters. As for LDA, it requires a full pass through the entire corpus at each iteration.

To fix this issue, we load documents in batches. For each batch t of size S, we compute the local parameters ϕ_t and γ_t, holding λ fixed. Then, we compute an estimate of λ related to that batch that we call $\tilde{\lambda}_t$ and we update λ as a weighted average between its earlier value and $\tilde{\lambda}_t$:

$$\lambda = (1 - \rho_t)\lambda + \rho_t\tilde{\lambda}_t,\ \rho_t = (\tau_0 + t)^{-s}$$

where ρ_t controls the importance of old values given the new ones. In particular:

- increasing the value of $\tau_0 \geq 0$ slows down the initial updates
- $s \in (0.5, 1]$ controls the rate at which old information is forgotten

We can also incorporate updates of α and η:

- $\alpha = \alpha - \rho_t\tilde{\alpha}_t$
- $\eta = \eta - \rho_t\tilde{\eta}_t$

The updates are affected by:

- $\tilde{\alpha}_t = [Hess^{-1}(l)][\nabla_\alpha l]$, where l is the sum of the ELBO of each document of the batch
- $\tilde{\eta}_t = [Hess^{-1}(l)][\nabla_\eta l]$
- $\phi_{d,n,k} \propto e^{\mathbb{E}_q[\ln \theta_{d,k}] + \mathbb{E}_q[\ln \beta_{k,w_n}]}$
- $\gamma_{d,k} = \alpha_k + \sum_n \phi_{d,n,k}$
- $\tilde{\lambda}_{t,k,n} = \eta_k + \frac{D}{|S|}\sum_{d=t\cdot S}^{(t+1)S-1} \phi_{d,n,k}$

Note that if the batch size $|S|$ is equal to one, the algorithm becomes online.

After the execution of the algorithm, each row d of the final document-topic matrix can be obtained using the Maximum Likelihood Estimation method over $q(\theta_d|\gamma_d)$. The same reasoning can be applied for topics, using MLE over $q(\beta_k|\lambda_k)$.

3.4 Hierarchical Dirichlet process (HDP)

HDP was proposed in [WPB11] as a way to overcome the problem of choosing the number of topics in LDA and derived methods while using an online variational inference algorithm. In particular, the number of topics is inferred from data.

Since this method uses the Dirichlet process, we need to introduce it first.

3.4.1 Dirichlet process (DP)

The Dirichlet process can be used as a generalization of the Dirichlet distribution which does not have a fixed number of topics.

Formally, it is defined as a two parameter distribution $DP(\alpha, H)$, where α is the concentration parameter and H is the base distribution.

To get an intuition on how it works, it can be viewed as a black-box which takes as input a continuous distribution H and produces a discrete distribution G_0 whose similarity with H is governed by α: as $\alpha \to \infty$, $G_0 = H$.

The procedure of generating G_0 can be described through the stick-breaking process. The basic idea is that you have a stick of unit length and you break it off. At each iteration i, you pick the remaining part of the stick and you break it off again and again. The entire process takes an infinite number of steps since you can always break off the remaining part of a stick, ending up to smaller and smaller pieces.

More formally:

$$\beta'_k \sim Beta(1, \alpha)$$

$$\beta_k = \beta'_k \prod_{l=1}^{k-1} (1 - \beta'_l)$$

where β'_k is the percentage of the remaining stick that is break off and β_k is the size of the broken part of the stick after step k. All β_k are then used as weights:

$$\phi_k \sim H$$

$$G(x) = \sum_{k=1}^{\infty} \mathbb{1}_{[x==\phi_k]}\beta_k$$

3.4.2 HDP nesting DPs

Using HDP instead of LDA or Online LDA requires the definition of a new model. First, we will describe the generative process. Then, we will analyse the steps of model inference.

3.4.2.1 Generative process

The model is composed of two layers:

- $\boldsymbol{G_d | G_0 \sim DP(\alpha, G_0)}$: each document d is described by its own DP G_d
- $\boldsymbol{G_0 \sim DP(\gamma, H)}$: each G_d is sampled from another DP G_0; in this way all G_d have the same atoms and they differ only in their weights

where H is a symmetric Dirichlet distribution over the vocabulary.

The stick-breaking construction of G_0 is the same as the one presented in Section 3.4.1. The one for each G_d is instead:

- $c_{d,k} \sim \mathit{Mult}(\beta)$
- $\pi'_{d,k} \sim \mathit{Beta}(1, \alpha)$
- $\pi_{d,k} = \pi'_{d,k} \prod_{l=1}^{k-1}(1 - \pi'_{d,l})$
- $G_j(x) = \sum_{k=1}^{\infty} \mathbb{1}_{[x==\phi_{c_{d,k}}]}\pi_{d,k}$

The generation of a word in position n is then:

- $z_{d,n} \sim \mathit{Mult}(\pi_d)$
- $\theta_{d,n} = \phi_{c_{d,z_{d,n}}}$
- $w_{d,n} \sim \mathit{Mult}(\theta_{d,n})$

The plate notation of HDP is represented in Figure 3.4.

3.4.2.2 Model inference

Due to the fact that this process requires an infinite number of steps, truncation is needed to view the generative process and the inference as algorithms. In particular, we set a

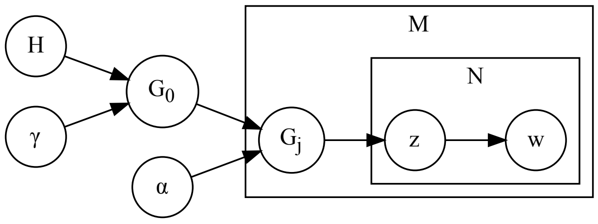


Figure 3.4: Graphical model representation of HDP.

maximum number of global topics K and a maximum number of local topics T. Note that a reasonable choice is to set $T \ll K$ since the number of topics needed for a single document is less than the number of topics required for the full corpus.

Our variational distribution is:

$$q(\beta', \pi', c, z, \phi) = q(\beta'|u, v)q(\pi'|a, b)q(c|\varphi)q(z|\zeta)q(\phi|\lambda)$$

Factorizing it results in:

- $q(c|\varphi) = \prod_d \prod_{t=1}^{T} q(c_{d,t}|\varphi_{d,t})$
- $q(z|\zeta) = \prod_d \prod_n q(z_{d,n}|\zeta_{d,n})$
- $q(\phi|\lambda) = \prod_{k=1}^{K} q(\phi_k|\lambda_k)$
- $q(\beta'|u, v) = \prod_{k=1}^{K-1} q(\beta'_k|u_k, v_k)$
- $q(\pi'|a, b) = \prod_d \prod_{t=1}^{T-1} q(\pi'_{d,t}|a_{d,t}, b_{d,t})$

where $q(\beta'_K = 1) = 1$ and $\forall d \; q(\pi'_{d,T} = 1) = 1$.

As we did for LDA and Online LDA we take derivatives of the ELBO with respect to each variational parameter. The results are both document-level updates obtained using coordinate ascent and corpus-level updates computed using gradient ascent. These updates can be performed in an online fashion, computing local updates only for a small part of documents for each iteration and then performing the corpus-level updates using a noisy estimation of the gradient. Since the updates and the steps required to obtain them are similar to previous methods, we deliberatively removed them from the description of HDP. However, the interested reader can find all the details at [WPB11].

Chapter 4

Network inference

Network inference is the process of inferring a set of links among variables that describe the relationships between data. In particular, in this chapter, we describe ARACNE, an algorithm introduced by [MNB$^+$06] for the reconstruction of Gene Regulatory Networks (GRNs).

4.1 ARACNE

ARACNE (Algorithm for the Reconstruction of Accurate Cellular Network) can describe the interactions in cellular networks starting from microarray expression profiles; the result is an undirected graph of the regulatory influences between genes.

This type of method arises from the need of having tools to separate artifacts from real gene interactions. In particular, ARACNE can be divided into three steps:

1. computation of Mutual Information (MI) between genes
2. identification of a meaningful threshold
3. Data Processing Inequality (DPI)

4.1.1 Computation of Mutual Information (MI) between genes

Mutual Information is a measure of relatedness between two random variables. For two discrete variables x and y, it is defined as:

$$MI(x, y) = H(x) + H(y) - C(x, y)$$

where $H(x)$ is the entropy of x and $C(x, y)$ is the joint entropy between x and y. Note that from the definition of Mutual Information, $MI(x, y) = 0$ if and only if $P(x, y) = P(x)P(y)$.

In our case, for each combination of genes x, y with M observations we want to obtain:

$$MI(\{x_i\}_{i=1,\dots,M}, \{y_i\}_{i=1,\dots,M})$$

Since microarray data is expressed by continuous variables, we have to use the differential entropy instead of the entropy and the joint differential entropy instead of the joint entropy. For this reason, an estimation of $p(x)$ using Kernel Density Estimation (KDE) is first needed:

$$\hat{p}(x) = \frac{1}{M} \sum_{i=1}^{M} K(x - x_i)$$

where i is an index which iterates through all observations of a gene x and $K(.)$ is a kernel. In particular, [MNB+06] proposes $K(.)$ to be a Gaussian kernel.

Finally, we compute the MI between two genes x, y using the formula:

$$\begin{aligned} MI(\{x_i\}, \{y_i\}) &= \iint \hat{p}(x, y) \log[\frac{\hat{p}(x, y)}{\hat{p}(x)\hat{p}(y)}] \, dx \, dy \\ &\approx \frac{1}{M} \sum_{i=1}^{M} \log[\frac{\hat{p}(x_i, y_i)}{\hat{p}(x_i)\hat{p}(y_i)}] \end{aligned}$$

4.1.2 Identification of a meaningful threshold

Since the MI cannot be zero by construction and there might be spurious dependencies between genes, we now want to find a threshold for which all pairs of genes with a MI under that value are considered independent.

To do so, we first randomly shuffle the expression of genes for each observation and we then compute the MI using this new dataset. This process is repeated several times, to obtain an approximate distribution of MIs under the condition of shuffled data.

We then set a p-value p, assuming that the first $1 - p$ percentage of MIs in the distribution of shuffled data is random noise. We then identify the biggest MI value in that group to be used as a threshold.

Given the MI values obtained in Section 4.1.1, we now set to zero the ones lower than the threshold.

4.1.3 Data Processing Inequality (DPI)

Finally, the DPI step prunes indirect relationships for which two genes are co-regulated by a third one but their MI is nonzero.

Given each possible triplet of genes (x, y, z), it checks if $MI(x, z) \leq \min[MI(x, y), MI(y, z)]$ and in that case it sets $MI(x, z) = 0$.

At the end, two genes are connected by an undirected link if and only if their final MI is greater than zero.

Chapter 5

Experiments

This chapter explains step-by-step our original work, built on top of the theory presented before.

Steps are grouped into sections, sorted by chronological order. The last section (Section 5.6) considers instead the scalability issues that we encountered and how we managed them.

5.1 Crawling

In order to collect data, we implemented a spider using Scrapy that given a particular domain, it downloads and store all HTML documents and the relative structure built upon the links that we encounter.

The application can be summarized with these steps:

1. first, the software starts from an URL defined by the user, putting it into a pool
2. if the pool is not empty, the application will get a link from it, starting its download. After getting the link, it is removed from the pool
3. if the content is a valid HTML document it is stored
4. all links of that webpage which are in the specified domain are stored. They are also put into the pool if they were not analyzed previously
5. loop to step 2 until there are no links left

The final result is a set of tuples `(url, connected_to, content)`, where `url` is the URL of a particular page, `connected_to` is its set of links and `content` is the content of the "<main>" tag. Scrapy allows saving these results in different formats, but we chose to save everything in CSV files in order to re-use them easily in the next phases. Note that we consider only what it is inside of the "<main>" tag, in order to store only the main content for each page.

Before using the scraped data, we performed a data preparation phase over it. During this step, we made a strong assumption about webpages: URLs with same path but different scheme correspond to the same page. For instance, `http://www.example.com` and `https://www.example.com` are considered links to the same resource.

Following this assumption, we then removed all duplicates from the "url" column of the CSV file: only the first occurrence of each URL is kept.

In order to use it as a dataset, we decided to scrape `corsi.unige.it` starting the procedure from `https://corsi.unige.it`. This decision was made for the following reasons:

- it is monolingual (pages are in Italian)
- we have prior knowledge of its structure
- we know that it has recurring patterns in its content

Note that in Section 5.5 we leverage the last point to evaluate the quality of the inferred network.

Thanks to the scraping started on 3 June 2020, we obtained a dataset of 20722 documents and 29286 unique terms (stop words excluded). After the preprocessing phase, the number of documents and words remained the same. These results are consistent with our prior knowledge of the website and with the assumption that we made before.

5.2 Similarity graph

After preprocessing we performed Topic Modeling using HDP. In this phase the content of each HTML document is parsed and only the raw text is kept to be used for inference. Common stop words are also removed.

Given the document-topic matrix, we then computed the Hellinger distance (see Section C.2) between each pair of documents (p, q):

$$HD(p,q) = HD(q,p) = \frac{1}{\sqrt{2}}\sqrt{\sum_i (\sqrt{p(i)} - \sqrt{q(i)})^2}$$

The final output can be viewed as a similarity graph represented through an adjacency matrix. Since each computed Hellinger distance d is always in the interval $[0, 1]$ by definition, the distance $1 - d$ could be used as the weight of the edge between each pair of documents.

One problem that we must deal with is the fact that webpages in a domain could have portions of HTML in common which are not relevant to the content of the page itself. For this reason, from now on we decided to keep two similarity graphs:

- a filtered version where words that appear in more than 50% of documents are not considered
- an unfiltered one

5.3 Weight binarization of the network

We now want to find a good threshold to obtain a sparse representation of the fully connected network built before.

Unfortunately, we cannot use the same approach of ARACNE (see Section 4.1) since we have to deal with probability distributions instead of samples, breaking the possibility of shuffling the data.

To overcome this issue, we built a scraper which samples randomly pages of `wikipedia.it` with replacement, using the URL `https://it.wikipedia.org/wiki/Speciale:PaginaCasuale`. The basic idea is that it is unlikely that a relatively small number of random pages share topics; the resulting Hellinger distance between them can be then considered noise. Furthermore, we must collect enough documents to have a meaningful HDP model.

For this reason, we decided to scrape 100 pages and to compute the two similarity graphs as previously done for `corsi.unige.it`. Since the number of Wikipedia webpages is greater than one million[1], the probability of sampling the same document more than once is relatively small. Thanks to this, it was possible to have a procedure that resembles the one presented in Section 4.1.2 which filters out spurious weights given a particular p-value.

This procedure was repeated 10 times with a p-value of 0.01. Results can be consulted in Figure 5.1. The averaged results were used to produce the two unweighted networks.

5.4 Data exploration

We then compared the structure of the website built through hyperlinks and the inferred networks. In this scenario, each link tag is considered an undirected and unweighted edge

[1]Source: `https://it.wikipedia.org/w/index.php?title=Speciale:Statistiche&action=raw`

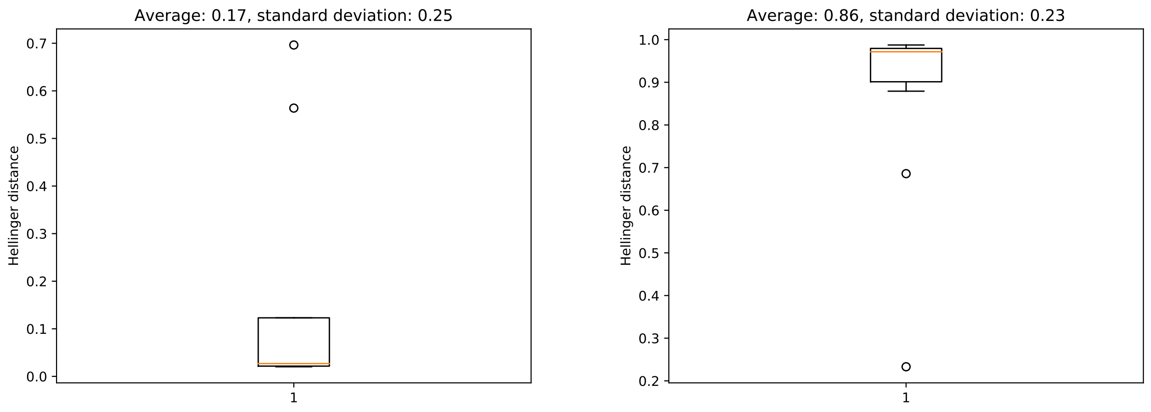


Figure 5.1: Boxplot of the computed thresholds for the filtered version (left) and the unfiltered one (right).

between the URL of the webpage in which it is found and the URL of its "href" attribute.

The following proprieties were analyzed:

- the degree distribution
- the clustering coefficient

where the clustering coefficient of a node is the number of possible triangles through that node normalized by the potential maximum number of them and the average clustering coefficient is just an average between all nodes of a graph. The histograms of the two proprieties expressed before can be seen in Figure 5.2.

Thanks to this analysis we concluded that the inferred networks and the website structure are graphs with different characteristics. In particular, the structure is sparser and less clustered than the inferred networks.

5.5 Evaluation of the results

Evaluating the quality of the result is not a trivial task.

Since we knew from Section 5.4 that the inferred networks are sparse, we focused our analysis only on the links between documents.

Thanks to the prior knowledge of the structure of the UniGe website, we knew that all courses provided by UniGe have a particular starting page and that they share a common tree-like structure. For instance, all courses have a page called "Orientamento" which have similar content everywhere. Figure 5.3 shows this pattern graphically.

In order to provide a score for evaluating if the inferred networks have a meaningful structure, we grouped pages by their similarities (e.g. all "Orientamento" webpages are in the same group) and we checked how many pages in the group are connected.

The results show that the unfiltered network recovers more than the 82% of the edges, while the filtered one identifies more than the 99% of them.

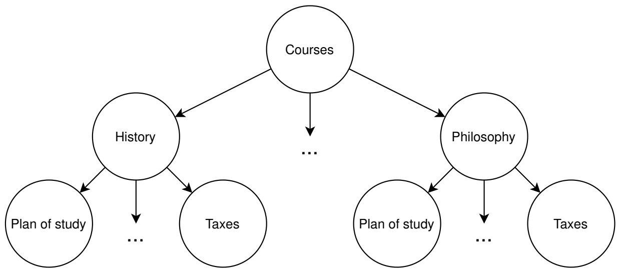


Figure 5.3: Subsampling of the UniGe website structure with titles translated to English.

5.6 Scaling issues

All source code is licensed under the GNU General Public License v3.0 and it can be found at `https://github.com/Davide95/msc_thesis/tree/master/code`. It is written in Python 3.7 and the external libraries used are reported in the relative `requirements.txt` file. The source code was also analyzed with “flake8” and “pylint” to improve readability and to spot bugs.

During the writing of the source code, we encountered some scalability issues related to the size of the datasets and the computational complexity of the algorithms used. This section contains a summary of the main strategies used to overcome this problem. All experiments were tested on a PowerEdge T630 server with the following characteristics:

- Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (2 sockets)
- 8x16GiB of RAM

5.6.1 NumPy

The majority of operations in our source code are applied to vectors and matrices. Modern CPUs provide SIMD (Single Instruction, Multiple Data) instructions like Intel® AVX-512 to speed up this kind of computations. In order to use them in the source code, we make use of a library called NumPy[2] which is interoperable between different hardware and computing platforms.

Furthermore, it allows us to have more numerical stability when dealing with huge matrices and vectors since it takes into account the limitations related to storing real numbers using finite-precision floating-point representations (for instance, using pairwise summation in its source code at numpy/core/src/umath/loops.c.src).

The interested reader can find more advantages of using NumPy reading [vCV11].

5.6.2 Numba

When vectorization is not enough due to the computational costs of the algorithms used, we compile portions of the Python code to machine code through Numba[3].

Thanks to this tool, portions of the software can:

- run at native machine code speed
- run without the involvement of the Python interpreter
- be parallelized using multithreading

In particular, parallelization is done releasing first the GIL (Global Interpreter Lock). The interested reader can find more details about it at `https://docs.python.org/3/glossary.html#term-global-interpreter-lock`.

A basic introduction to how Numba works can be found at [LPS15].

5.6.3 Multiprocessing

We are not able to use NumPy nor Numba where the code is not numerically orientated. Furthermore, it is not possible to release the GIL in some portions of the code due to the limitations of the imported libraries. For these reasons, there are situations in which multiprocessing is used.

[2]`https://numpy.org/`
[3]`https://numba.pydata.org/`

In particular, we have to use multiprocessing instead of multithreading during the parsing of HTML documents explained in Section 5.2. This choice allows us to scale using every core of our machine but at the cost of increasing the memory consumption of the application. To have a trade-off between computational costs and memory issues, we decided to add two parameters that can be tweaked.

5.6.4 Optimizing memory usage

In Python, it is not possible to deallocate manually memory blocks; a GC (Garbage Collector) is instead responsible to remove data from memory when needed.

To help its work, we try to avoid using the Jupyter Notebook when we found high memory usage. The reason is mainly that it tends to abuse the use of the global scope for storing variables and it keeps additional references to objects. Furthermore, we split the code into different steps, each step restrained in a function. Between each step, we manually force the GC to collect unused resources.

Finally, we avoid using lists when they do not have a fixed size. The reason is that frequent append operations in a short time lead in multiple reallocations of memory which could not be deallocated in time and a memory shortage might happen. In particular, the problem is that lists in Python are not linked lists but they use exponential over-allocation every time they lack space. To make a comparison, they have the same mechanism as ArrayList in Java. For a more detailed explanation of how memory management works for lists in Python, please refer to its source code at Objects/listobject.c.

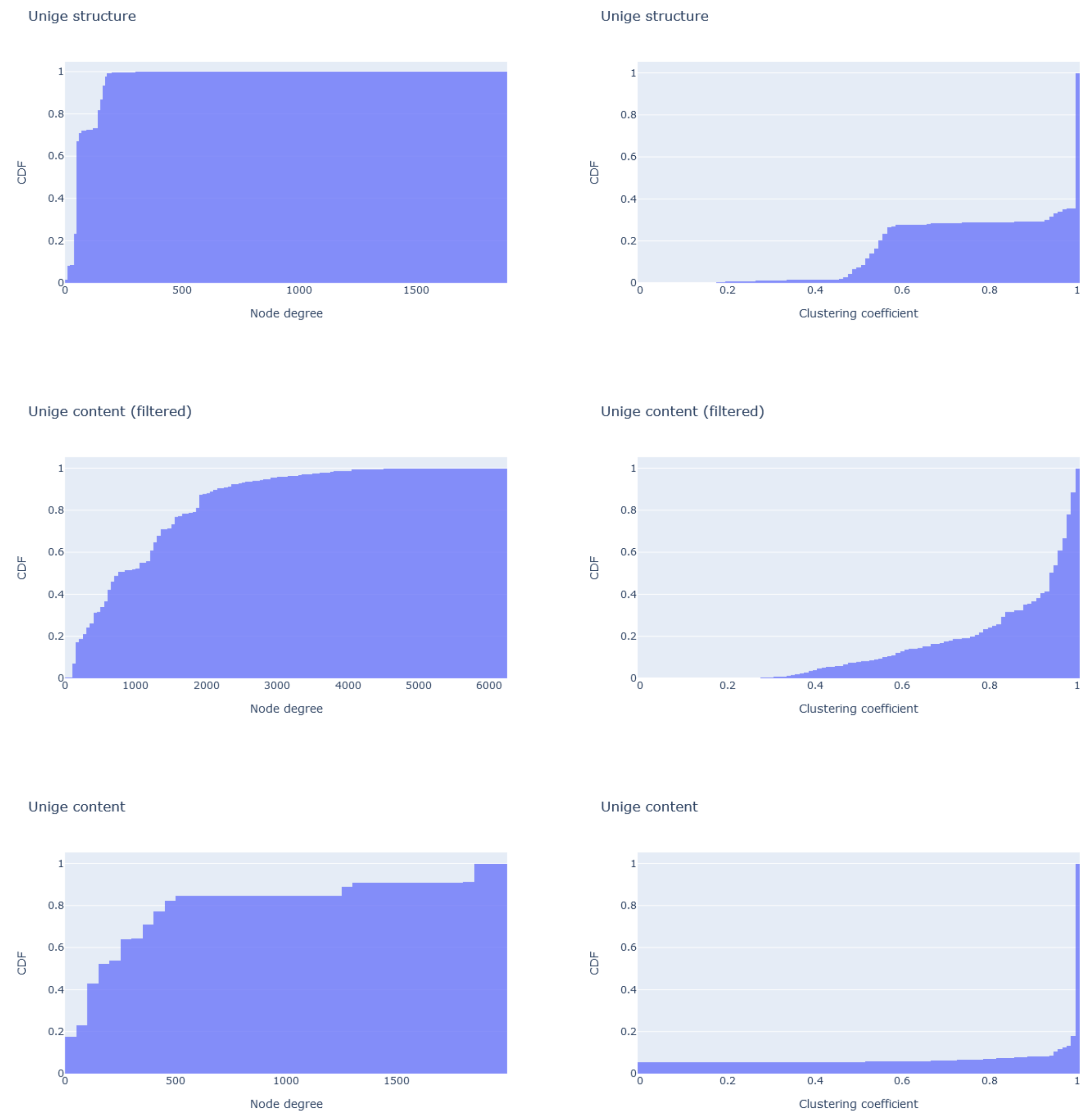


Figure 5.2: Cumulative distribution function of the node degrees (first column) and the clustering coefficients (second column). The first row is referring to the structure of the website, while the second and third row show the results respectively from the filtered similarity graph and the unfiltered one.

Chapter 6

Conclusions and future work

The main purpose of this work was to infer similarities between documents in a collection, obtaining a network in which vertices are documents and edges express similarities of the content. In particular, we worked with scraped websites showing that for our data it is possible to construct a graph of relations between thousands of webpages without scaling issues. Note that being a website is not a requirement and our pipeline can be used in the future also for datasets which are only a collection of documents.

An extension of our pipeline might be to consider the semantic tags, assigning different weights to the text inside them. An example could be to consider each title inside a "<h1>" tag twice important than text outside it, counting the words twice during the Bag-of-Words conversion.

Due to limited time, we dealt with monolingual websites, but we propose an approach to overcome this issue. The problem can be divided into two steps: an automatic identification of the language of a portion of text and the production of an algorithm to have the same Bag-of-Words representation for each concept expressed in different languages. Since we are working with web data, it is possible to retrieve the language of the content of each webpage by looking at the Content-Language entity header (see [FR14]) or the lang global attribute (see the HTML 5.1 W3C Recommendation[1]). Note that the second approach is preferable when "lang" is provided since it handles the situation in which at least one webpage is composed of sections with different languages. When both options are not available, a language identification model (see fastText resources[2]) could be used as a failover. In order to have different languages with a common Bag-of-Words representation, we could translate each document to a pre-defined language and then compute the BoW format for each document as previously done. The lack of pre-trained models covering a

[1] https://www.w3.org/TR/html51/dom.html#the-lang-and-xmllang-attributes

[2] https://fasttext.cc/docs/en/language-identification.html

wide range of languages requires to find alternative solutions to the problem of translation. we suggest to look for already trained multilingual word embeddings which are aligned in a common space (see fastText resources[3]) to use them to find an association between words expressing the same concept in different languages. The assumption we made in the last scenario is that words in different languages share the same meaning and for this reason different word vectors of different languages differ only on the basis of the vector space in which they are represented. The interested reader may refer to [JBMG18] for further analysis of the topic.

Preliminary results suggest that the inferred network can model relations between topics. Further analyses could be made with different datasets in order to empirically verify what we observed with our data. In particular, we propose Cora[4] since it provides a network of connections between documents that can be used as a ground truth.

[3] `https://fasttext.cc/docs/en/english-vectors.html`
[4] `https://linqs.soe.ucsc.edu/data`

Appendix A

Neural Networks

The goal of this appendix is to give a simple albeit incomplete introduction of Neural Networks for those readers who do not have a solid background on them but still they are interested in Chapter 2.

A.1 Gradient descent

Given a minimization problem $min_w f(w), w \in \mathbb{R}^D$, it is possible to find a local minimum through gradient descent.

First, we compute the gradient $\nabla f(w)$ and we decide a point w_0 in which the algorithm has to start. If the problem is not convex, choosing a different w_0 could lead to a different solution. Since gradients point in the direction of fastest increase, we take small steps towards the opposite direction in order to reach a minimum.

The step size is controlled by a hyperparameter λ: values too small for a particular problem increase the time needed to obtain a good solution and it can lead to a local minimum; values too big prevent convergence.

The final equation is:

$$w_{i+1} = w_i - \lambda \nabla f(w_i)$$

Each step is run iteratively until a stopping criterion is met. If the algorithm is stopped after I iterations, the solution for the minimization problem is w_I. Common sense stopping criterions could be:

- stop after a prefixed number of iterations
- stop when the results are not getting better for the last I steps

- stop when $f(w_i) - f(w_{i+1}) < \epsilon$, with $\epsilon > 0$ small

Note that the parameter λ could be different at each iteration:

$$w_{i+1} = w_i - \lambda_{i+1} \nabla f(w_i)$$

For instance, we could have $\lambda_i = \frac{\lambda}{i}$ in order to take smaller and smaller steps until convergence.

A.1.1 Gradient descent in action

Suppose to have a dataset $X \in \mathbb{R}^{N \times D}, y \in \mathbb{R}^N$.

What we want to do is linear regression, whose model is obtained resolving:

$$\min_{w \in \mathbb{R}^D} \frac{1}{2N} \sum_{i \in [1,N]} (w^T x_i - y_i)^2$$

The derivative $\frac{1}{2N} \sum_i (w^T x_i - y_i)^2 dw$ is:

$$\frac{1}{2N} \sum_{i \in [1,N]} 2x_i(w^T x_i - y_i) = \frac{\sum_{i \in [1,N]} x_i(w^T x_i - y_i)}{N}$$

Therefore:

$$w_{n+1} = w_n - \frac{\lambda}{N} \sum_{i \in [1,N]} x_i(w_i^T x_i - y_i)$$

A.2 Mini-batch gradient descent

There are situations in which using gradient descent is not computationally feasible. For instance, imagine that the problem explained in Section A.1.1 have a number of points so huge that a solution cannot be obtained in feasible times.

Mini-batch gradient descent is an extension of gradient descent which assumes that samples of $|M|$ elements of the dataset are a noisy representative of the entire data. At each step, M will be re-sampled from X.

In that case, the minimization presented in Section A.1.1 can be rewritten as:

$$\min_{w \in \mathbb{R}^D} \frac{1}{2|M|} \sum_{x_i \in M} (w^T x_i - y_i)^2$$

A.3 Linear Neural Networks

The linear regression exposed in Section A.1.1 can be viewed as a Neural Network which has D inputs as a first layer and a single neuron as output connected to all previous ones.

The graphical representation of it can be seen in Figure A.1.

The vector w is simply the set of all weights. Starting from the equation presented in Section A.1.1:

$$w_{n+1} = w_n - \frac{\lambda}{N} \sum_{i \in [1,N]} x_i(w^T x_i - y_i)$$

each weight w_i can be parallelly updated:

$$w_{(n+1)j} = w_{nj} - \frac{\lambda}{N} \sum_{i \in [1,N]} x_{ij}(w^T x_i - y_i)$$

In this example, the cost function is the square loss, but there is not any particular requirement when dealing with Neural Networks.

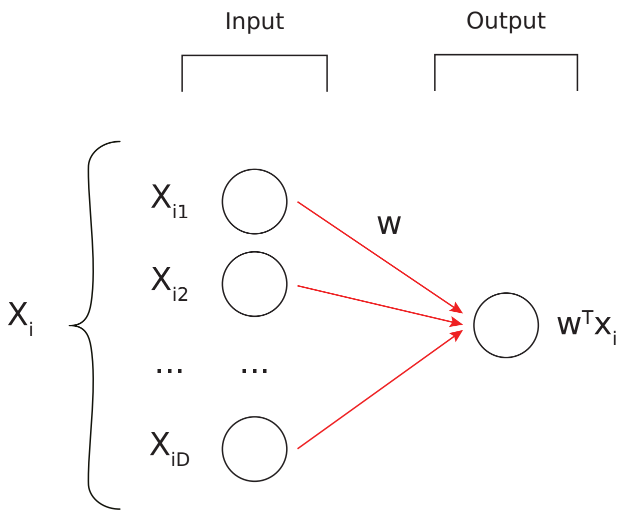


Figure A.1: A graphical representation of a linear Neural Network.

A.4 Activation functions

In each node, a non-linear function could be applied at the data before moving the output to the next layer. For the sake of simplicity, we are going to present a model similar to the previous one, with the difference that a $\tanh(\cdot)$ function is applied to the output node:

$$\min_{w\in\mathbb{R}^D} \frac{1}{2N} \sum_{i\in[1,N]} [\tanh(w^T x_i) - y_i]^2$$

To resolve it with gradient descent, we have to apply the chain rule. Since $\tanh(z)dz = 1 - tanh^2(z)$:

$$\frac{1}{2N} \sum_{i\in[1,N]} [\tanh(w^T x_i) - y_i]^2 dw = \frac{1}{2N} \cdot \sum_{i\in[1,N]} 2[\tanh(w^T x_i) - y_i] \cdot [1 - tanh^2(w^T x_i)] \cdot x_i$$

$$w_{n+1} = w_n - \frac{\lambda}{N} \sum_{i\in[1,N]} [\tanh(w^T x_i) - y_i][1 - tanh^2(w^T x_i)]x_i$$

Other activation functions can be used instead of $\tanh(\cdot)$. The one used in the models presented in Chapter 2 is the softmax $\sigma(z), z \in \mathbb{R}^V$:

$$\sigma(z,j) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$\sigma(z) = \begin{bmatrix} \sigma(z,1) \\ \sigma(z,2) \\ \ldots \\ \sigma(z,V) \end{bmatrix}$$

A.5 Multilayer Neural Networks

Adding intermediate layers (usually called hidden layers in the literature) to a linear Neural Network is useless since the product of linear components is again linear. As an example, construct a model with the following characteristics:

- $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{N\times 2}}$: 2 nodes as input
- $\boldsymbol{y} \in \mathbb{R}^{\boldsymbol{N}}$: 1 node as output
- 2 nodes in the hidden layer, fully connected to the two other layers

Its graphical representation can be seen in Figure A.2 and the relative minimization function is:

$$\min_{M,w} \frac{1}{2N} \sum_{i\in[1,N]} [w^T(x_i M) - y_i]^2 \text{ with } M \in \mathbb{R}^{2\times 2}, w \in \mathbb{R}^2$$

which is again linear regression but more computationally expensive.

On the opposite, stacking more layers while using a non-linear activation function leads to an enriched hypothesis space. Learning weights in a Deep Neural Network (a Neural Network with multiple hidden layers) requires using the backpropagation algorithm.

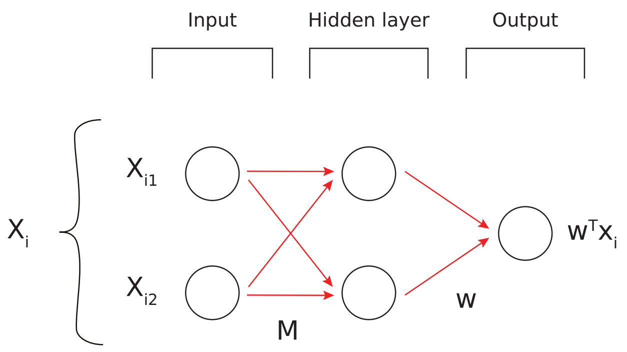


Figure A.2: A graphical representation of a Neural Network with an hidden layer.

A.5.1 Chain rule

The chain rule states that given $\hat{y} = f(g(x))$ its derivative is:

$$\frac{d\hat{y}}{dx} = \frac{f(g)}{dg} \cdot \frac{g(x)}{dx}$$

That formula can be seen as a Neural Network with one node as input, one as output and one in the hidden layer. Its graphical representation can be viewed in Figure A.3.

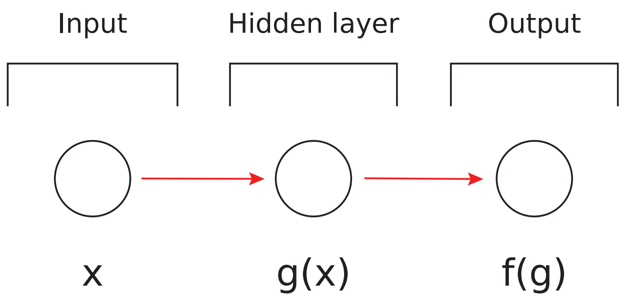


Figure A.3: A graphical representation of the chain rule.

A.6 Backpropagation

To better understand backpropagation, imagine having a fully connected Neural Network with:

- D nodes as input
- J nodes in the hidden layer with tan as activation function
- 1 node as output with tan as activation function

This network has weights $M \in \mathbb{R}^{D\times J}$ between the first and the second layer, and weights $w \in \mathbb{R}^{J}$ connecting the second layer to the third.

The minimization function is

$$\min_{M,w} \frac{1}{2} \sum_{i\in[1,N]} E(X,y)_i = \min_{M,w} \frac{1}{2} \sum_{i\in[1,N]} \{\tanh[w^T \tanh(X_i M)] - y_i\}^2$$

The final goal is to obtain:

- $\dfrac{dE_i}{dw}$
- $\dfrac{dE_i}{dM_{dj}}$

In order to make the notation as light as possible, the function $\tanh(\cdot)$ is intended to be applied element-wise. Moreover, the intermediate result $\tanh(MX_i)$ is denoted as h_i and the symbol $\circ$ represents the element-wise product between two vectors.

To compute the partial derivatives, we can start from the output layer and then use the intermediate calculations to compute the partial derivatives of the previous layers. In particular:

$$\frac{dE_i}{dy_i} = \frac{1}{2} \cdot 2\{\tanh[w^T \tanh(MX_i)] - y_i\} = \tanh(w^T h_i) - y_i$$

$$\frac{dE_i}{dw} = \frac{dE_i}{dy_i}\frac{dy_i}{dw} = \frac{dE_i}{dy_i}\{1 - \tanh^2[w^T \tanh(MX_i)]\} \tanh(X_i M)$$

$$\frac{dE_i}{dh_i} = \frac{dE_i}{dy_i}\frac{dy_i}{dh_i} = \frac{dE_i}{dy_i}\{1 - \tanh^2[w^T \tanh(MX_i)]\} w$$

$$\frac{dE_i}{dM_{dj}} = [\frac{dE_i}{dh_i} \circ \{\mathbb{1} - \tanh^2(X_i M)]\}_j X_{id}$$

The power of this technique is that it is possible to re-use values calculated for a particular partial derivative to lower the computational cost of computing the others.

The reasoning can be extended for multiple nodes as output.

A.7 References

Details are omitted and the topic is discussed only in summary fashion, although the reference paper is provided for the interested reader: [Ron14].

Appendix B

Probability distributions

Over all probability distributions used in this document, some of them are not usually part of the canonical background of a computer scientist. We decided to present them in this appendix in order to have an overview for who has never seen them before.

B.1 Beta distribution

The Beta distribution is a family of continuous probability distributions. It is defined in the range $[0, 1]$ and it has two parameters $\alpha, \beta > 0$.

Since α and β are concentration parameters, they induce more and more sparsity as they approach the value of zero. If $\alpha = \beta = 1$, the probability density function is uniform over $[0, 1]$. As their values grow, the distribution tightens over its expectation. All these observations can be seen empirically in Figure B.1.

Due to its characteristics, it is suited to model probabilistic outcomes (for example, to describe how confident we are about the fact that a coin is loaded or not).

B.1.1 Probability density function

Given a random variable $X \sim Beta(\alpha, \beta)$, the probability density function is defined as:

$$f_X(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

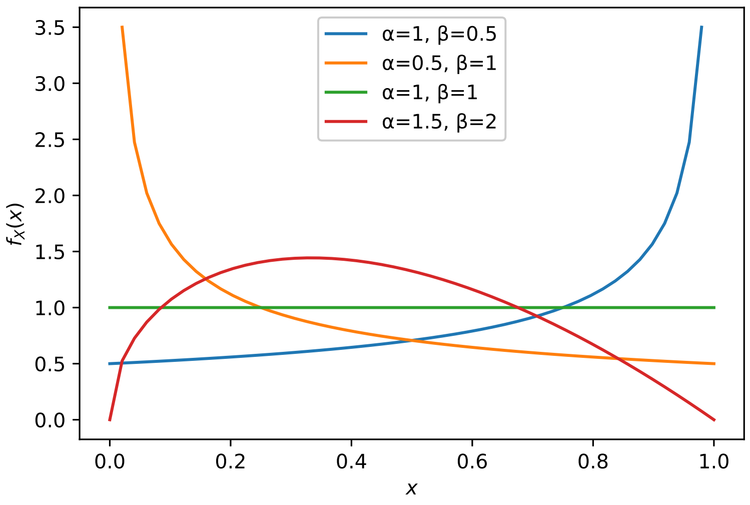


Figure B.1: Example of the probability density function of a Beta distribution varying its parameters.

$B(\alpha, \beta)$ is the Beta function and it acts as a normalization constant to ensure that:

$$\int_0^1 f_X(x)dx = 1$$

B.2 Dirichlet distribution

The Dirichlet distribution is a family of multivariate distributions. It is parametrized by K parameters, where K is the number of random variables for which it is described.

It can be seen as a generalization of the Beta distribution: a Dirichlet distribution of 2 parameters $Dir(\alpha_1 = a, \alpha_2 = b)$ is a Beta distribution $Beta(\alpha = a, \beta = b)$. All considerations and constraints about the parameters of the Beta distribution presented previously (Section B.1) still holds for the Dirichlet distribution.

B.2.1 Probability density function

Given a multivariate random variable $X \sim Dir(\alpha_1, \alpha_2, \ldots, \alpha_K)$, the probability density function is defined as:

$$f_X(x_1, x_2, \ldots, x_K; \alpha_1, \alpha_2, \ldots, \alpha_K) = \frac{1}{B(\alpha_1, \alpha_2, \ldots, \alpha_K)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

$B(\alpha_1, \alpha_2, \ldots, \alpha_K)$ is the multivariate generalization of the Beta function presented in Section B.1.

Setting $Dir(\alpha) \coloneqq Dir(\alpha_1, \alpha_2, \ldots, \alpha_n)$, a simplified notation of the probability density function is used:

$$f_x(x_1, x_2, \ldots, x_K; \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha - 1}$$

The support is constrained by:

- $x_i \in (0, 1), 1 \leq i \leq K$
- $\sum_{i=1}^{K} x_i = 1$

B.2.2 Graphical intuition

The support of a Dirichlet distribution with K parameters lies in a $K - 1$ simplex. If $K = 3$, the simplex is a triangle and can be represented graphically (see Figure B.2).

All values of the probability density function $f_X(x_1, x_2, x_3) : \Re^3 -> \Re$ are associated with a specific color given their intensity. Bigger values in the probability density function are represented with lighter colors and vice versa.

Vertices of the simplex represent the situations in which $x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, $x = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ or $x = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. If $x = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$, we are at the center of the simplex. Given an area of the triangle lighter than others, the probability of obtaining a vector in that interval during sampling is higher and vice versa.

Suppose we are in a supermarket and we are interested in the kind of purchases consumers make. We define three categories: "food", "technology" and "stationery". A strong preference over one category is represented modelling the probability density function with higher values towards one of the vertices. Sampling from a Dirichlet distribution as the one represented in the left simplex in Figure B.2 will tend to give us users who do not have a strong preference over one category. The opposite situation (for example, $\alpha_1 = 0.1$, $\alpha_2 = 0.1$, $\alpha_3 = 0.1$) will otherwise give us customers who strongly prefer one of the categories (for example. $x = \begin{bmatrix} 0.8 & 0.15 & 0.05 \end{bmatrix}$) while users without propensities will be much rare. Finally, if the probability density function is uniform over all the support (right simplex in Figure B.2) there is no tendency of choosing a particular combination of categories over others.

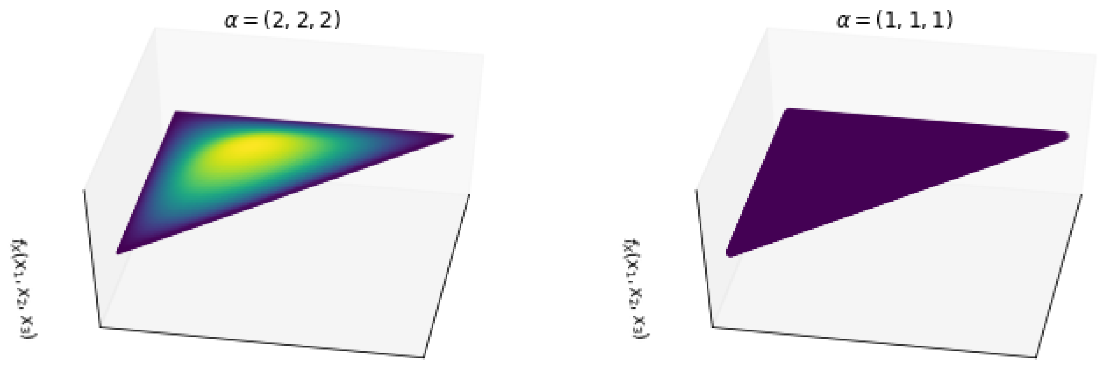


Figure B.2: Example of the probability density function over a simplex of a Dirichlet distribution varying its three parameters $\alpha = (\alpha_1, \alpha_2, \alpha_3)$. The lighter the color, the higher the probability density function.

B.3 Categorial distribution

The categorical distribution is a generalized Bernoulli distribution in the case of categorical random variables.

In the literature about Topic Modeling, it is wrongly used with the name of Multinomial distribution. To keep the same notation as the papers in the bibliography, we will say that $X \sim Mult(p)$ means that X is distributed as Categorical. Note that a Categorical distribution $Mult(p)$ is a special case of a Multinomial distribution $Mult(n, p)$ in which $n = 1$.

Each p_i of the parameter $p = (p_1, p_2, \ldots, p_K)$ represents the probability of observing i by sampling from the distribution.

B.3.1 Probability mass function

Given a random variable $X \in Mult(p)$, the probability mass function is defined as:

$$P(x == i) = p_i$$

B.4 References

The interested reader may refer to [Gup11], [BNJ03] and [Seb11] to use them as references.

Appendix C

Probability distributions closeness

Some methods were developed to assess closeness between different probability distributions (see Figure C.1).

In this appendix, we cover the ones which are used in the thesis.

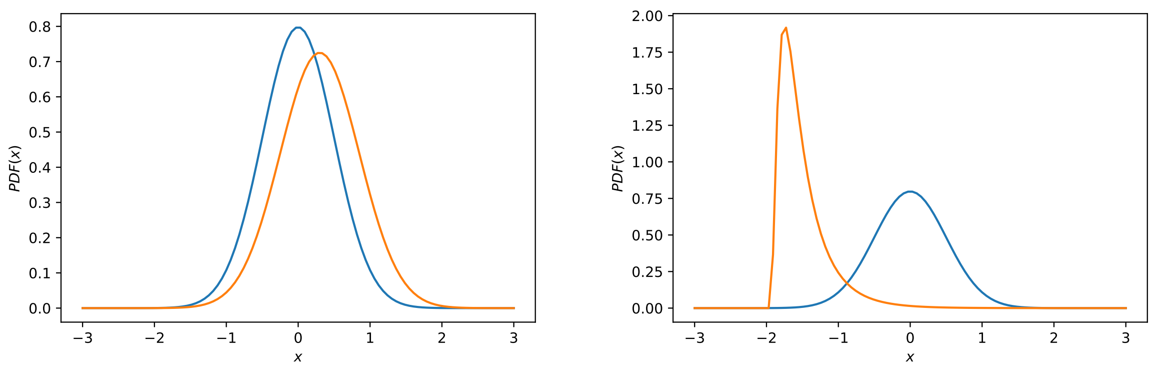


Figure C.1: On the left plot the probability density function of two close distributions, the opposite on the right plot.

C.1 Kullback–Leibler divergence

Given two probability distributions $p(x)$ and $q(x)$, we define the KL divergence as:

$$KL(p\|q) = \int_x p(x) \log_2 \frac{p(x)}{q(x)} = \mathop{\mathbb{E}}_p [\log_2 \frac{p(x)}{q(x)}]$$

If the two distributions are discrete, it can be rewritten as:

$$KL(p\|q) = \sum_i p(i) \log_2 \frac{p(i)}{q(i)}$$

Suppose we have two Categorical distributions c_1 and c_2 with parameters $p_{c_1} = (0.1, 0.2, 0.7)$ and $p_{c_2} = (0.3, 0.5, 0.2)$, respectively.

Given the formula presented above, we can compute the KL divergence:

- $KL(c_1\|c_2) = 0.1 \log_2 \frac{0.1}{0.3} + 0.2 \log_2 \frac{0.2}{0.5} + 0.7 \log_2 \frac{0.7}{0.2} \approx 0.85$
- $KL(c_2\|c_1) = 0.3 \log_2 \frac{0.3}{0.1} + 0.5 \log_2 \frac{0.5}{0.2} + 0.2 \log_2 \frac{0.2}{0.7} \approx 0.77$

C.1.1 Proprieties

From the previous example, it can be seen that the KL divergence is not symmetric: $KL(p\|q) \neq KL(q\|p)$.

Also, note that $KL(p\|p) = \int_x p(x) \log_2 \frac{p(x)}{p(x)} = \int_x p(x) \log_2(1) = \int_x p(x) \cdot 0 = 0$ and $KL(p\|q) \geq 0$.

C.2 Hellinger distance

The Hellinger distance has some advantages over the Kullback–Leibler divergence:

- **it is symmetric**: $HD(p, q) = HD(q, p)$
- **it has an upper bound**: $0 \leq HD(p, q) \leq 1$

Given two discrete probability distributions p and q, the Hellinger distance is defined as:

$$HD(p, q) = HD(q, p) = \frac{1}{\sqrt{2}} \sqrt{\sum_i (\sqrt{p(i)} - \sqrt{q(i)})^2}$$

Note that in the literature the Hellinger distance is defined in different ways (e.g. with or without $\frac{1}{\sqrt{2}}$).

C.3 References

The interested reader may refer to [Joy11], [Lin91] and [Hel09] for further analysis of the topic.

Appendix D

Variational Inference

Variational inference is a method that approximates probability densities through optimization.

Denoting x as the observations and z as the hidden variables, we want to compute the posterior distribution $p(z|x)$.

In particular, we want to find a $q(z|v)$ over a family of densities which is as similar as possible to $p(z|x)$ using the Kullback–Leibler divergence as a measure of closeness:

$$\begin{aligned} KL(q||p) &= \mathop{\mathbb{E}}_q [\ln \frac{q(z|v)}{p(z|x)}] \\ &= \mathop{\mathbb{E}}_q \ln[q(z|v)] - \mathop{\mathbb{E}}_q \ln[p(z|x)] \\ &= \mathop{\mathbb{E}}_q \ln[q(z|v)] - \mathop{\mathbb{E}}_q \ln[\frac{p(z,x)}{p(x)}] \\ &= \mathop{\mathbb{E}}_q \ln[q(z|v)] - \{\mathop{\mathbb{E}}_q \ln[p(z,x)] - \mathop{\mathbb{E}}_q \ln[p(x)]\} \\ &= \mathop{\mathbb{E}}_q \ln[q(z|v)] - \{\mathop{\mathbb{E}}_q \ln[p(z,x)] - \ln[p(x)]\} \\ &= \mathop{\mathbb{E}}_q \ln[q(z|v)] - \mathop{\mathbb{E}}_q \ln[p(z,x)] + \ln[p(x)] \end{aligned} \tag{D.1}$$

Note that $q(z|v)$ does not depend on the observed data.

To do so, we start by computing $p(x)$:

$$\begin{aligned}
p(x) &= \ln[p(x)] \\
&= \ln[\int p(x,z)dz] \\
&= \ln[\int \frac{q(z|v)}{q(z|v)} p(x,z)dz] = \ln[\int q(z|v)\frac{p(x,z)}{q(z|v)}dz] \\
&= \ln[\mathop{\mathbb{E}}_{q} \frac{p(x,z)}{q(z|v)}]
\end{aligned}$$

The Jensen's inequality says that if a function f is concave, $f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$. Since the logarithm is a concave function, we can state that:

$$\ln[\mathop{\mathbb{E}}_{q} \frac{p(x,z)}{q(z|v)}] \geq \mathop{\mathbb{E}}_{q} \ln[\frac{p(x,z)}{q(z|v)}]$$

$$\ln[\mathop{\mathbb{E}}_{q} \frac{p(x,z)}{q(z|v)}] \geq \mathop{\mathbb{E}}_{q} \ln[p(x,z)] - \mathop{\mathbb{E}}_{q} \ln[q(z|v)]$$

Given that $\mathop{\mathbb{E}}_{q}\{\ln[p(x,z)] - \ln[q(z|v)]\} = -E_q\{\ln[p(x,z)] - \ln[q(z|v)])\}$ and the fact that $\ln[p(x)]$ is a constant that does not depend on q, maximizing the lower bound of $p(x)$ is equivalent to minimizing the Kullback–Leibler divergence between q and p (see Equation D.1 to a comparison). From now, we refer to this lower bound as ELBO.

D.1 Mean field variational inference

In order to have a tractable problem, these distributions must have the characteristic of removing the mutual dependence between the hidden variables:

$$q(z_1, ..., z_m|v) = \prod_{j=1}^{m} q(z_j|v_j)$$

Since we want to maximize ELBO, we first check that it is a concave function and then we take the partial derivatives of it and set them equal to zero. Then, we solve the problem iteratively for each variable of interest using coordinate-ascent optimization until convergence. Note that the algorithm converges to a local maximum and not a global one.

D.2 References

The interested reader is advised to read [BKM17] in conjunction with this in order to have an overall idea of the topic.

Bibliography

[BKM17] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Feb 2017. URL: `http://dx.doi.org/10.1080/01621459.2017.1285773`, `doi:10.1080/01621459.2017.1285773`.

[Ble11] David Blei. Probabilistic topic models. In *Proceedings of the 17th ACM SIGKDD International Conference Tutorials*, KDD '11 Tutorials, New York, NY, USA, 2011. Association for Computing Machinery. `doi:10.1145/2107736.2107741`.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. URL: `http://jmlr.org/papers/v3/blei03a.html`.

[CB09] Jonathan Chang and David Blei. Relational topic models for document networks. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 81–88, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL: `http://proceedings.mlr.press/v5/chang09a.html`.

[Dum04] Susan T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, 2004. URL: `https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105`, `arXiv:https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/aris.1440380105`, `doi:10.1002/aris.1440380105`.

[FR14] R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, RFC Editor, June 2014. URL: `https://www.rfc-editor.org/rfc/rfc7231.txt`.

[Gup11] Arjun K. Gupta. *Beta Distribution*, pages 144–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. `doi:10.1007/978-3-642-04898-2_144`.

[HBB10] Matthew Hoffman, Francis R. Bach, and David M. Blei. Online learning for latent dirichlet allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 856–864. Curran Associates, Inc., 2010. URL: http://papers.nips.cc/paper/3902-online-learning-for-latent-dirichlet-allocation.pdf.

[Hel09] E. Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 136:210–271, 1909. URL: http://eudml.org/doc/149313.

[JBMG18] Armand Joulin, Piotr Bojanowski, Tomas Mikolov, and Edouard Grave. Improving supervised bilingual mapping of word embeddings. *CoRR*, abs/1804.07745, 2018. URL: http://arxiv.org/abs/1804.07745, arXiv:1804.07745.

[Joy11] James M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-04898-2_327.

[Lin91] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2833157.2833162.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL: https://arxiv.org/abs/1301.3781.

[MNB$^+$06] Adam A. Margolin, Ilya Nemenman, Katia Basso, Chris Wiggins, Gustavo Stolovitzky, Riccardo Dalla Favera, and Andrea Califano. ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinform.*, 7(S-1), 2006. doi:10.1186/1471-2105-7-S1-S7.

[NvNvdG19] Malvina Nissim, Rik van Noord, and Rob van der Goot. Fair is better than sensational: Man is to doctor as woman is to doctor. *CoRR*, abs/1905.09866, 2019. URL: https://arxiv.org/abs/1905.09866, arXiv:1905.09866.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: `https://www.aclweb.org/anthology/D14-1162`.

[Ron14] Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014. URL: `https://arxiv.org/abs/1411.2738`, `arXiv:1411.2738`.

[Seb11] George A. F. Seber. *Multinomial Distribution*, pages 882–884. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. `doi:10.1007/978-3-642-04898-2_388`.

[SW15] Adriaan M. J. Schakel and Benjamin J. Wilson. Measuring word significance using distributed representations of words. *CoRR*, abs/1508.02297, 2015. URL: `https://arxiv.org/abs/1508.02297`, `arXiv:1508.02297`.

[vCV11] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, 2011.

[WPB11] Chong Wang, John W. Paisley, and David M. Blei. Online variational inference for the hierarchical dirichlet process. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 752–760. JMLR.org, 2011. URL: `http://proceedings.mlr.press/v15/wang11a/wang11a.pdf`.

License