Exercises
Computational Intelligence Lab
SS 2020

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Prof. Dr. Thomas Hofmann
Web http://da.inf.ethz.ch/cil

Series 6, Solutions (LDA and Word Embeddings)

Problem 0 (LDA Theory):

i. $p(z|d)$ (the probability of a topic given a document) is learned only for documents on which the model is trained. This might be inconvenient, and makes pLSA not a well-defined generative model. Indeed, to generate a new document we need to be able to sample topic weights $u_i = (u_{1i}, \dots, u_{Ki})^\top$. pLSA has no good way to provide a new u_i, thus only heuristics are possible: for instance, a new document is "folded in" and EM is re-run (holding the old parameters fixed) to estimate the topic distribution parameter for this new document. Also, it is important to note that in pLSA the number of parameters grows linearly with corpus.

ii. Consider a simplified setting:

- total of N words in our dictionary, and we pick (with replacement) L_i words to form document d_i;
- order of the words does not matter: document summarized in vector x_i, where x_{ij} counts occurrence of word j in document i;
- probability of word j in document i is π_{ij}.

Simple combinatorics *give* (i.e. not an assumption) rise to the multinomial distribution.

$$p(x_i|\pi_i) = \frac{L_i!}{\prod_j x_{ij}!} \prod_{j=1}^{N} \pi_{ij}^{x_{ij}}, \quad x_i \sim \text{Multi}(\pi_i).$$

If word probabilities follow a Dirichlet, i.e. $\pi_i \sim \text{Dir}(\alpha_i)$, then something magic happens to the posterior:

$$\begin{aligned} p(\pi_i|x_i, \alpha) &\propto p(x_i|\pi_i)p(\pi_i|\alpha_i) \\ &= \left(\frac{L_i!}{\prod_j x_{ij}!} \prod_{j=1}^{N} \pi_{ij}^{x_{ij}} \right) \left(\frac{1}{B(\alpha_i)} \prod_{j=1}^{N} \pi_{ij}^{\alpha_{ij}-1} \right) \propto \prod_{j=1}^{N} \pi_{ij}^{x_{ij}+\alpha_{ij}-1}. \end{aligned}$$

$\Rightarrow$ posterior also going to be Dirichlet, with parameter $x_i + \alpha_i$. This makes it possible to derive approximate inference algorithms to solve (i.e. to approximate posteriors for) LDA.

iii. Advantages of LDA:

- Given the larger number of parameters, pLSA might be more susceptible to overfitting than LDA;
- One can see the Dirichlet prior as a regularizer on the topics distribution;
- model is theoretically sound and motivated (conjugacy, exponential family, approximate inference etc.);

But..

- If your corpus is fixed, not clear one would need a generative model (even though it might be elengant);
- pLSA is far more easy to implement, used more often in industry;
- pLSA and LDA often achieve comparable performance;
- the hyperparameter α, if chosen in a wrong way, can make LDA way worse that pLSA.

For a more detailed explanation, please check `https://link.springer.com/content/pdf/10.1007/s10791-010-9141-9.pdf`

Problem 1: Latent Vector Model

i. English grammar implies $p(w|w') > p(w'|w)$. However, the proposed model, we have

$$\log(p(w|w')) = \langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + \beta = \langle \mathbf{x}_{w'}, \mathbf{x}_w \rangle + \beta = \log(p(w'|w)).$$

Hence $p(w|w')$ always equals $p(w'|w)$. A solution might be to use an asymmetric model:

$$p(w|w') = \langle \mathbf{x}_w, \mathbf{y}_{w'} \rangle + \beta.$$

ii. Recall the $\log$-likelihood function is

$$L(\mathbf{x}) = \sum_{n=1}^{N} \sum_{\triangle \in \mathcal{I}} \log p(w_{n+\triangle}|w_n)$$

where $\mathcal{I}$ is the window. Suppose that the window size is 3. According to this likelihood, we need to find an embedding $\mathbf{z}$ such that

$$\arg\max_{\mathbf{z}} \langle \mathbf{z}, \mathbf{x}_{\text{the}} \rangle + \langle \mathbf{z}, \mathbf{x}_{\text{on}} \rangle + \langle \mathbf{z}, \mathbf{x}_{\text{chair}} \rangle.$$

You can similarly solve the problem for window size 4.

Window size	predicted word (horse, mat, or chair)
3	**Solution: chair**
4	**Solution: mat**

Problem 2 (Introduction to SGD):

i. A straight-forward computation concludes the result:

$$\mathbf{E}_r \left[\nabla f_r(\mathbf{w}) \right] = \sum_{i=1}^{n} P(r=i) \nabla f_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w}).$$

ii. The negative gradient gives a linear approximation to the direction of steepest descent of a function. Hence, a simple way to optimize a function is to "follow the gradients" for short steps of length γ. We illustrate convergence in the deterministic case using the function $f(w) = w^2/2$. With a step $\gamma = 0.5$ the algorithm reads $w \leftarrow w - 0.5w = 0.5w$. Starting from any $w_0 \in \mathbb{R}$, the algorithm converges to the minimizer $w^* = 0$.

iii. The intuition behind it does not change if stochastic gradient are used, but the algorithm will be sometimes slower iteration-wise, since the optimizer is "confused" by the noise. Yet, using stochastic gradients (if n is really big) results in a superior per-iteration complexity (true gradient is a sum over all stochastic gradients).

Problem 3 (Negative sampling):

i. The gradient with respect to any $\mathbf{x}_v$ of the normalization constant $\log \left(\sum_{w' \in \mathcal{V}} \exp[\langle \mathbf{x}_{w'}, \mathbf{y}_w \rangle] \right)$ is

$$\frac{\exp(\langle \mathbf{y}_w, \mathbf{x}_v \rangle)}{\sum_{w' \in \mathcal{V}} \exp(\langle \mathbf{y}_{w'}, \mathbf{x}_v \rangle)} \mathbf{y}_w.$$

This gradient includes a summation over the discionary $\mathcal{V}$. In most cases, there are 20,000 or 1 million words which makes this sum prohibitively large if one wants to run the SGD algorithm (let alone gradient descent!) which needs to be able to compute many small updates inexpensively (see previous exercise).

ii. Note that, for any $f : \mathbb{R}^d \to \mathbb{R}$ and $x \in \mathbb{R}^d$, $\nabla_x \ln \sigma(f(x)) = \frac{1}{\cancel{\sigma(f(x))}} \cancel{\sigma(f(x))} (1 - \sigma(f(x))) \nabla_x f(x)$. Thus, the gradient with respect to $\mathbf{x}_v$ of the negative sampling objective is is

$$\sum_{(v,w) \in \triangle^+} (1 - \sigma(\langle \mathbf{x}_v, \mathbf{y}_w \rangle)) \mathbf{y}_w + \sum_{(v,w) \in \triangle^-} (\sigma(\langle \mathbf{x}_v, \mathbf{y}_w \rangle) - 1) \mathbf{y}_w.$$

First, note that this computation does not depend on $|\mathcal{V}|$ and should be fast enough to allow for cheap gradient updates. Second, the overall gradient is completely decoupled into a sum, thus one can update the objective in sampled batches of the data.

Problem 4 (GloVe and SGD):

1) a) Apply the chain rule of gradients,

$$\nabla_{\mathbf{x}_i} \mathcal{H} = \sum_{j:\, n_{ij}>0} 2(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{y}_j,$$
$$\nabla_{\mathbf{y}_j} \mathcal{H} = \sum_{i:\, n_{ij}>0} 2(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{x}_i.$$

b) $\mathcal{H} = \sum_{ij} \mathcal{H}_{ij}$ where $\mathcal{H}_{ij} := f(n_{ij})(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij})^2$. Since the gradient is linear, it factorizes over this sum as $\nabla \mathcal{H} = \sum_{ij} \nabla \mathcal{H}_{ij}$, enabling us to apply SGD. Moreover, it makes sense to compute the stochastic gradient with respect to the relevant embedding vectors, which is:

$$\nabla_{\mathbf{x}_i} \mathcal{H}_{ij} = 2(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{y}_j,$$
$$\nabla_{\mathbf{y}_j} \mathcal{H}_{ij} = 2(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{x}_i.$$

2) Plug in the specific weight function:

$$\mathcal{H} = \sum_{i,j:\, n_{ij}>0} (m_{ij} - \langle \mathbf{x}_i, \mathbf{y}_j \rangle)^2 = \sum_{i,j:\, n_{ij}>0} \left(m_{ij} - (\mathbf{X}^\top \mathbf{Y})_{ij}\right)^2.$$

the minimization of which w.r.t. $\mathbf{X}, \mathbf{Y}$ is a problem of matrix completion.

3) We have

$$\nabla_{\mathbf{x}_i} \mathcal{H} = \sum_{j:\, n_{ij}>0} 2f(n_{ij})(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{y}_j,$$
$$\nabla_{\mathbf{y}_j} \mathcal{H} = \sum_{i:\, n_{ij}>0} 2f(n_{ij})(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{x}_i.$$

$$\nabla_{\mathbf{x}_i} \mathcal{H}_{ij} = 2f(n_{ij})(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{y}_j,$$
$$\nabla_{\mathbf{y}_j} \mathcal{H}_{ij} = 2f(n_{ij})(\langle \mathbf{x}_i, \mathbf{y}_j \rangle - m_{ij}) \mathbf{x}_i.$$

Problem 5-6 (Coding):

The solution files are in the public git repository:

`https://github.com/dalab/lecture_cil_public/tree/master/exercises/ex6`

Hint for Step 3 of the GloVe implementation): you can use PCA provided by scikit-learn, e.g. `http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_3d.html` *or any other library.*