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, March 26-27, 2020 (LDA and Word Embeddings)

Problem 0 (LDA Theory):

Read carefully the lecture slides for pLSA and LDA, then try to answer the following (pretty difficult) questions:

i. What are the limitations of pLSA that led to the design of LDA?

ii. LDA is a Bayesian method where a fundamental role is played by the Dirichlet prior. Why is this prior so important for the theory behind the LDA model? Hint: look up "conjugacy" on a Bayesian statistics reference (or Wikipedia).

iii. What are the disadvantages of LDA over pLSA? Is LDA actually an improvement over pLSA?

Problem 1 (Latent Vector Model):

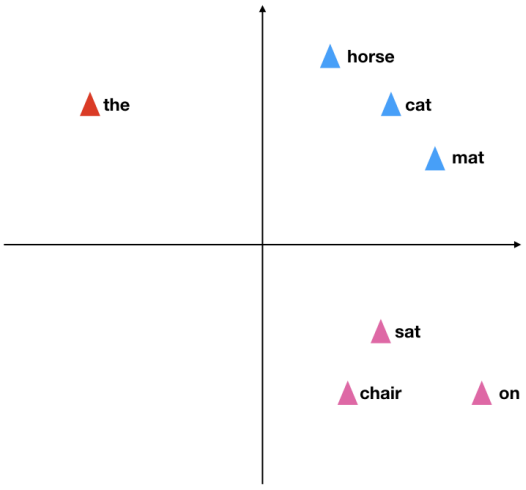
Consider the log-bilinear model for word embeddings seen in the lecture:

$$\log p(w|w') = \langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + \beta, \tag{1}$$

where $p(w|w')$ is the probability that w occurs in context window of w'.

i. Suppose w' is an adjective and w is a noun. Why we do expect that $p(w|w') > p(w'|w)$ for context window of consisting of the next word? Why can not the model of Eq. (1) capture this? How we can refine the model to solve this issue?

ii. Consider the following 2D embeddings of words (i.e. $\mathbf{x}_w$ in Eq. (1)).



Given the above embeddings, we want to complete the following sentence using one of words "horse", "mat", or "chair".

The cat sat on the ...

According to the skip-gram model with window size 3 and 4, which words are more probable to complete the sentence?

Window size	predicted word (horse, mat, or chair)
3	.....
4	.....

Problem 2 (Introduction to SGD):

The purpose of this exercise is to get you a bit familiar with (stochastic) gradient descent. Consider minimization of function f that is an average of functions $\{f_1, \ldots, f_n\}$ as

$$f(\mathbf{w}) = \frac{1}{n} \sum_{i=1} f_i(\mathbf{w}).$$

Stochastic gradient descent (SGD) optimizes f through the following recurrence:

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma \nabla f_r(\mathbf{w}),$$

where r is selected uniformly at random from $\{1, \ldots, n\}$ (independently from $\mathbf{w}$).

i. Prove that $\nabla f_r(\mathbf{w})$ is an unbiased estimator of $\nabla f(\mathbf{w})$, namely

$$\mathbf{E}_r \left[\nabla f_r(\mathbf{w})\right] = \nabla f(\mathbf{w}).$$

ii. Assume for simplicity that we optimize with the full gradient $\nabla f(\mathbf{w})$ instead of the the stochastic gradient. What is the intuition behind the algorithm? Prove convergence on $f(w) = w^2/2$ with a step $\gamma = 0.5$ using deterministic gradients.

iii. Do you expect the behaviour outlined in the last point to change when using the estimator $\nabla f_r(\mathbf{w})$? When/why is it desirable to use stochastic gradients?

Problem 3 (Negative sampling):

Suppose we want to learn an embedding from a text of lenght T. Let v and w be two words in the vocabulary $\mathcal{V}$. Consider the following model for the probability of observing v given the context of w:

$$\log p(v|w) = \langle \mathbf{x}_v, \mathbf{y}_w \rangle + c,$$

using embedding $\mathbf{y}_w$ and $\mathbf{x}_v$.

i. To infer the embeddings using a maximum likelihood approach, we need to optimize the following function:

$$L(\mathbf{x}, \mathbf{y}) = -\sum_{t=1}^{T} \sum_{\triangle \in \mathcal{I}} \langle \mathbf{x}_{w^{t+\triangle}}, \mathbf{y}_{w^t} \rangle - \log \left(\sum_{v \in \mathcal{V}} \exp[\langle \mathbf{x}_v, \mathbf{y}_{w^t} \rangle] \right). \tag{2}$$

where $\mathcal{I} = \{-R, \ldots, -1, 1, \ldots R\}$ defines the context window. Explain why (Stochastic) Gradient Descent on the above objective is not practical: what is the bottleneck?

ii. In the lecture, you have seen the alternative objective

$$L(\triangle_+, \triangle_-; \mathbf{x}, \mathbf{y}) = \sum_{(v,w) \in \triangle_+} \log \sigma(\langle \mathbf{x}_v, \mathbf{y}_w \rangle) + \sum_{(v,w) \in \triangle_-} \log \sigma(-\langle \mathbf{x}_v, \mathbf{y}_w \rangle), \tag{3}$$

where $\triangle_+, \triangle_-$ are the sets of positive and negative examples (see lecture). Is this objective easier to optimize with SGD? How would you do it?

Problem 4 (GloVe and SGD):

Recall the GloVe objective that consists in weighted least squares fit of log-counts, written as

$$\mathcal{H}(\theta; \mathbf{N}) = \sum_{i,j} f(n_{ij}) \Big(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\langle \mathbf{x}_i, \mathbf{y}_j \rangle}_{\text{model}} \Big)^2 ,$$

where $\langle \mathbf{x}_i, \mathbf{y}_j \rangle = \log \tilde{p}_\theta(w_i | w_j)$. Note that we ignore here the bias terms for simplicity.

i. Assume $f(n_{ij}) = 1$ for all ij, and write $m_{ij} := \log n_{ij}$.

   a) Derive the derivative (gradient) of $\mathcal{H}$ with respect to the embedding vector $\mathbf{x}_i$, and the same for $\mathbf{y}_j$.

   b) Derive a stochastic gradient of $\mathcal{H}$, given by the contribution of just the term for (i, j) as part of the sum, again with respect to the embedding vector $\mathbf{x}_i$, and the same for $\mathbf{y}_j$.

ii. Show that GloVe with $f(n_{ij}) := \begin{cases} 1 & \text{if } n_{ij} > 0, \\ 0 & \text{otherwise,} \end{cases}$ solves the *matrix completion* problem

$$\min_{\mathbf{X}, \mathbf{Y}} \sum_{(i,j) \,:\, n_{ij} > 0} \big(m_{ij} - (\mathbf{X}^\top \mathbf{Y})_{ij} \big)^2 .$$

iii. Analogous to part 1), derive the gradient and the stochastic gradient of $\mathcal{H}$ with respect to $\mathbf{x}_i$ and $\mathbf{y}_i$ for a generic weighting function f.

Problem 5 (Project on sentiment classification):

In this assignment, we will use word embeddings as one possible approach to build our own text classifier system. The text sentiment classification task is the second of the three project tasks proposed. Here, we introduce the problem and competition rules.

The task of this competition is to predict whether a tweet message used to contain a positive or negative smiley, :) or :(, by considering only the remaining text.

Submission system environment setup:

1. The data and the template code are available at

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

   The kaggle web page for the competition can be found here. To join the competition, create a kaggle account using your `.ethz.ch` email address.

   https://inclass.kaggle.com/c/cil-text-classification-2019.

2. Download the provided dataset `train_pos.txt`, `train_neg.txt`, `test_data.txt`, and the submission example `sampleSubmission.csv` from the competition web page.

To submit your solution to the online evaluation system, we require you to prepare a ".csv" file of the same structure as `sampleSubmission.csv`. The order of predictions is arbitrary, but make sure the tweet ids and predictions match. Your submission is evaluated according to the classification error of your predictions (the number of misclassified tweets).

Working with the Twitter data:

We provide a large set of training tweets, one tweet per line. All tweets in the file `train_pos.txt` (and the `train_pos_full.txt` counterpart) used to have positive smileys, those of `train_neg.txt` used to have negative smileys. Additionally, the file `test_data.txt` contains 10'000 tweets without any labels, each line numbered by the tweet-id.

Your task is to predict the labels of these tweets, and upload the predictions to kaggle. Your submission file for the 10'000 tweets must be of the form `<tweet-id>, <prediction>` (see `sampleSubmission.csv`).

Note that all tweets have already been pre-processed so that all words (tokens) are separated by a single whitespace. Also, the smileys (labels) have been removed.

Classification via Word-Vectors. For building a good text-classifier, it is crucial to have a good feature representation of the input text. Here we start using the word vectors (word embeddings) for each word in the given tweet. For simplicity, we construct the feature representation of the entire text by simply averaging the word vectors.

Below is an example of the solution pipeline:

1. **Compute word embeddings:** Load the training tweets given in `train_pos_full.txt` and `train_neg_full.txt` (or a suitable subset depending on the RAM available), and construct a vocabulary, the list of words appearing at least 5 times in the subset chosen.

   Compute the GloVe word embeddings for each word from the vocabulary (see previous exercise).

2. **Construct features for the training texts:** Load the training tweets and the built GloVe word embeddings. Using the word embeddings, construct a feature representation of each training tweet by averaging the word vectors over all words from the tweet.

3. **Train a linear classifier:** Train a linear classifier (e.g. logistic regression or SVM) on your features constructed, using the `scikit learn` library. Recall that the labels indicate if a tweet used to contain a :) or :(smiley.

4. **Prediction:** Predict labels for all tweets in the test set.

5. **Submission, evaluation:** Submit your predictions to kaggle, and verify the obtained misclassification error score. (You can also use a local separate validation set to get faster feedback on the quality of your solution). Try to tune your system to achieve the best evaluation score.

Extensions: Naturally there are many ways to improve your solution, both in terms of accuracy and computation speed. More advanced techniques can be found in the recent literature.

Problem 6 (GloVe Implementation):

In this part, you will implement the GloVe models and train your own word vectors with stochastic gradient descent. We provide a set of tokenized tweets for which you have to learn the word embeddings.

1) Construct the co-occurrence matrix $\mathbf{N} = (n_{ij}) \in \mathbb{R}^{|\mathcal{V}| \cdot |\mathcal{C}|}$ where n_{ij} is the number of occurrences of the word $w_i \in \mathcal{V}$ in the context of $w_j \in \mathcal{C}$.

2) Implement the GloVe algorithm using Stochastic Gradient Descent. To do so, you can fill in the implementation of the cost and gradient functions in

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

   Once you are done, test your implementation by running on the data provided in the Text Sentiment Classification task, and tune the best-step-size parameters and dimensionality to achieve a good performance.

3) The word vectors typically capture strong linguistic regularities. For example, vector operations on the embeddings $w_{Paris} - w_{France} + w_{Italy}$ results in a vector that is very close to w_{Rome}, and $w_{king} - w_{man} + w_{woman}$ is close to w_{queen}. Your task is as follows:

   a) find some similar interesting configurations of embedded words from tweets,

   b) perform PCA to project the word vectors to a 2-dimensional space, and check whether the linguistic features are learned for the word embeddings.