INTRODUCTION TO TEXT MINING – INF 582

Lab 4: Word embeddings – unsupervised document classification

Antoine Tixier, Konstantinos Skianis and Michalis Vazirgiannis
February 3, 2017

1 Description and requirements

In this fourth lab session, we will play with word embeddings and see how they can be applied to unsupervised document classification. Among others, we will make use of Python's scikit-learn and gensim libraries.

You will also need to download the freely available 300-dimensional word vectors learned by Google (with word2vec) on the Google News data set (see "pre-trained word and phrase vectors" section here: https://code.google.com/archive/p/word2vec/). As the file equates to 1.5GB in size, you can ask one of the TAs to give it to you via a USB thumb drive if the download takes too much time. This file contains embeddings for more than 3 million unique words and phrases, but today, we will only be using a subset of it (once loaded in memory, the full file takes a lot of RAM).

2 Experimenting with word embeddings

First, we will get familiar with some properties of word embeddings by performing some operations manually. The steps below are implemented either fully or in part in the `main` script available under the `code` directory. Fill the gaps whenever it is needed.

- Load the word vectors and compute the cosine similarity in the embedding space between semantically close words (e.g., "man" and "woman") and between unrelated words. What do you observe?

- Project the word vectors into a lower-dimensional space using PCA and visualize some regularities on the first two principal directions. You can use sklearn's `PCA` function and its `fit.transform` method, and the custom `plot_points` function. What do you observe?

- Compute the cosine similarity of the two sentences "Kennedy was shot in Texas" and "The President was killed in Dallas" in the traditional *vector space* ("bag-of-words"). What do you observe? Using this time the cosine similarity of the centroids of the sentences in the *embedding space*, what do you observe? Note that the two sentences have no word in common.

3 Word embeddings for text classification

Averaging or summing (element-wise) the vectors of the words in a sentence or document is not the ideal way to go from word-word similarity to a similarity (or distance) between pieces of text. A better idea is by using the well-known Earth Mover's Distance[1], which extends the notion of distance between single elements to that of distance between sets of elements. This approach was illustrated by Kusner et al. (2015)[2] in the context of NLP (more precisely, unsupervised document classification). Other natural applications arise in information retrieval, for example. Kusner et al. called their distance the Word Mover's Distance (WMD). Even though it was only recently introduced, it is already used in practice[3]. Last year, the WMD was ported to the famous genism module, and we will make use of this implementation today.

More precisely, the WMD for two documents d and d' is defined as the minimum cumulative Euclidean distance that needs to be traveled in the embedding space to send all the words of document d to document d'. Computing the WMD is equivalent to solving a transportation problem that can be formulated as follows:

$$\min_{T \geq 0} \sum_{i,j}^{n} T_{i,j} . \left\| x_i - x_j \right\|_2$$

Where x_i is the embedding of word i in document d, x_i is the embedding of word j in document d', n is the size of the vocabulary and $T_{i,j}$ is a flow matrix indicating how much of word i in d travels to word j in d'. Solving the problem comes down to computing T given the constraints that the outgoing flow from a given word cannot exceeds its weight. Here, word weights are defined as their normalized counts. Specialized solvers for this linear programming optimization task have been developed[4]. However, complexity remains high: $O(m^3 \log m)$ where m is the number of words in the two documents.

As in the original WMD paper, we will use a Knn classifier[5] to categorize documents. For each example in the test set, the Knn classifier computes the distances between the example and all the instances in the training set in order to identify the k nearest neighbors of the example. Then, majority voting (the most frequent labels in the set of neighbors) is used to generate the prediction.

The procedure can be broken down as follows. Again, some of the steps are yours to fill:

- load documents from the 20NewsGroup data set using the interface provided by sklearn. For simplicity, we will only use documents from two different categories, making our classification task *binary*.

[1] http://robotics.stanford.edu/~rubner/papers/rubnerIjcv00.pdf
[2] http://www.jmlr.org/proceedings/papers/v37/kusnerb15.pdf
[3] http://tech.opentable.com/2015/08/11/navigating-themes-in-restaurant-reviews-with-word-movers-distance/
[4] http://www.cs.huji.ac.il/~werman/Papers/ICCV2009.pdf
[5] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

- implement a 4-fold cross-validation evaluation scheme for your Knn classifier, and compare multiple values of k. Since Knn requires many distance computations, and the WMD is quite expensive to compute, we will randomly sample a few observations in the training and test sets at each fold.

- compare the WMD distance with that of the cosine similarity between TF-IDF vectors. Keep in mind that the WMD is a distance between documents (the smaller, the better) while for cosine similarity, it is the opposite.

3 References

Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. Q. (2015). From Word Embeddings To Document Distances. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 957-966).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).