ADVANCED LEARNING FOR TEXT AND GRAPH DATA

MASTER DATA SCIENCE - MVA

Lab 5: Word embeddings – unsupervised document classification

Antoine Tixier, Konstantinos Skianis and Michalis Vazirgiannis
March 8, 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 use Python's `scikit-learn` and `gensim`'s word2vec (https://radimrehurek.com/gensim/models/word2vec.html) libraries.

You will need to download the publicly available 300-dimensional "Google News" word vectors from the "pre-trained word and phrase vectors" section here: https://code.google.com/archive/p/word2vec/. As the file is 1.5GB in size, it is better to download it in advance. As a last resort, you can ask one of the TAs to give it to you via a USB thumb drive at the beginning of the session. This binary file contains embeddings for more than 3 million unique words and phrases, learned with word2vec from a corpus of more than 100 billion words by Mikolov et al. (2013).

We will be working on a subset of the well-known 20 Newsgroup data set, which contains approximately 20,000 newsgroup documents partitioned equally over 20 different categories, like electronics, sports, politics… We will access the data through scikit's `fetch_20newsgroups()` function.

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.py` script available under the `code` directory. Fill the gaps wherever it is needed.

- Load the data, clean the documents and construct a list of lists of tokens to be passed to genism's `build_vocab()` method. Then, load the Google News word vectors corresponding only to the words in our vocabulary. This is a trick to avoid having to load all the vectors into memory (it would require 5-6 GB of RAM). 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? Similarly, perform some operations on concepts and interpret the results.

- Project the word vectors into a lower-dimensional space using PCA and visualize some regularities on two-dimensional maps (e.g., dim1-dim2). You can use sklearn's `PCA` function and its `fit.transform` method, and the custom `plot_points` function. What do you observe? Keep in mind that the picture you get is somewhat 'distorted' (low dimensionality, some information loss) – thus, do not expect perfect results.

- 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 unsupervised 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 (Rubner et al. 2000), which *extends the notion of distance between single elements to that of distance between sets of elements*. This metric was recently used in the context of NLP (document classification) by Kusner et al. (2015). Other natural applications arise in information retrieval, for example. In NLP, the Earth Mover's Distance is called the Word Mover's Distance (WMD). While quite recent, the WMD is already used in the industry[1]. Last year, it was ported to the famous genism module (`wmdistance()` method), 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'. The documents are represented as normalized bag of words vectors, that is, with entries $c_i / \sum_{k=1}^{n} c_k$, where c_k is the count of word k in the document, and n is the size of the vocabulary (unique words in the collection).

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_j 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 with the following constraints:

[1] http://tech.opentable.com/2015/08/11/navigating-themes-in-restaurant-reviews-with-word-movers-distance/

$\sum_{i=1}^{n} T_{i,j} = c_j / \sum_{k=1}^{n} c_k$: "the incoming flow to word j cannot exceed the weight of word j, or said differently, a word cannot receive more than its weight"

$\sum_{j=1}^{n} T_{i,j} = c_i / \sum_{k=1}^{n} c_k$: "the outgoing flow from word i cannot exceed the weight of word i, or put differently, a word cannot send more than its weight"

These constraints ensure that d is entirely transformed into d'. All the entries of T are positive as the flow always go the same direction (we do not allow reverse traveling).

Specialized solvers for this linear programming optimization task have been developed (Pele and Werman 2009). However, complexity remains high: $O(m^3 \log m)$, where m is the number of words in the two documents. The WMD can thus be considered an expensive metric.

As in the original WMD paper, we will use a Knn classifier[2] 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.

We will compare the performance of the WMD against that of cosine similarity between tfidf vectors in predicting the labels of the documents. Because the WMD is an expensive metric, we only run our experiments on a small subset of the collection. We implement a 4-fold cross validation scheme, and we test multiple numbers of nearest neighbors. The code can be found in the `k_fold_cv.py` script that should be called from the command prompt or the shell. Some warnings may be issued about genim depending on the version you are using; they should be ignored. Note that this script will parallelize the four folds over the cores of your machine (should work both on Windows and Unix). Parts of the code have been blanked out and are yours to fill. Fill them, and run the script. Then, go back to the `main.py` file, load the results, compute performance and interpret the scores.

There are some user parameters that you can tune in the `k_fold_cv.py` script.

- n_doc: total number of documents that should be used for the experiment. The more, the better, but the longer it will take. The current value of 60 runs in a couple of minutes on an i7 machine with four cores.
- th: threshold (in number of words) above which the longer documents are truncated (to speed-up computations). The current value is 300.
- ks: list of integers with the number of nearest neighbors to try (note that odd values are used to implicitly take care of tie-breaking). Current list: [1,3,5,7,11,17]

[2] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

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., 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).

Pele, O., & Werman, M. (2009, September). Fast and robust earth mover's distances. In *Computer vision, 2009 IEEE 12th international conference on* (pp. 460-467). IEEE.

Rubner, Y., Tomasi, C., & Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, *40*(2), 99-121.