ADVANCED LEARNING FOR TEXT AND GRAPH DATA

MASTER DATA SCIENCE - MVA

Lab 5: Word embeddings

Fragkiskos Malliaros, Antoine Tixier and Michalis Vazirgiannis
March 3, 2016

1 Description

In this fifth and last lab, we will play with word embeddings and see how they can be applied to text classification.

2 Experimenting with word embeddings

A sample (of size 24,500) of the freely available 300-dimensional word vectors learned by Google's word2vec on the Google News data set (see "pre-trained word and phrase vectors" section here) is available under the data directory as `word_vectors_lab5.csv`. Even though we will only be using a subset today, it is good to know that the original Google file offers embeddings for more than 3 million words, and can be downloaded directly into Python thanks to the gensim library (gensim also offers a very nice suite of tools for learning and using word embeddings).

The steps below are implemented either fully or in part in the `lab_5.py` file 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?

- For visualization purposes, project the word vectors into a lower-dimensional space (e.g., of dimension 50) using PCA. You may use sklearn's `PCA` function and its `fit.transform` method (see here),

- Plot the embeddings of specific words (e.g., "France", "Paris", "Germany", "Berlin") on the first two PCA dimensions and perform basic vector operations. 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 BOW space. What do you observe? Using this time the cosine similarity of the centroids of the word vectors of each sentence, what do you observe?

3 Word embeddings for text classification

In what follows, we will focus on a recent application of word embeddings to document categorization and information retrieval using a variant of the earth mover distance, the so called Word Mover's Distance. Even though it was only recently introduced, it is already used in practice. For this part you will need to install the pyemd Python module. Make sure you have the latest version of numpy and Anaconda (in case it applies). For Windows users, also make sure that you have a C++ compiler for Python 2.7.

The WMD for two documents d and d' is defined as the minimum cumulative distance that needs to be traveled to send the words of document d to the words of document d' in the word embeddings space. 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, n is the size of the vocabulary and $T_{i,j}$ is a transformation square 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 incoming flow equals the outgoing flow (d should be entirely transformed into d').

Using the pyemd Python module, the WMD between two documents can simply be computed with: `emd(doc_1_norm, doc_2_norm, D)` where `doc_1_norm` is the normalized BOW representation of document 1 (1 whenever a feature is present in the document, 0 else, divided by the number of features), and D is the Euclidean distance between all the features (symmetric array). By feature here we mean a unique non stopword (a member of the vocabulary).

As in the original paper, we will use a Knn classifier to categorize upcoming documents. The procedure can be broken down as follows. Again, some of these 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,

- split the documents into training and testing sets. Since Knn requires computing the distance between each testing example and all the documents in the training set (and the WMD is quite expensive to compute), we will only use very small train and test sets. In practice, advanced optimization is required to ensure scalability (see for instance this blog post),

- after cleaning the train and test sets, implement your Knn classifier. You can compare the WMD distance with the cosine similarity between centroids as your baseline. For each example in the test set, compute the distances (WMD and your baseline) to all the documents in the training set. In each case, retain the k nearest neighbors, and use majority

vote (the most frequent labels in the set of neighbors) as your prediction. Keep in mind that the WMD is a distance between documents (the smaller it is, the closer the documents) while for cosine similarity (if you use it as your baseline) it is the opposite.

- Compare performance

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).