INF5820, fall 2016
Assignment 3: Document Classification with Word Embedding Models

November 9, 2016

Deadline 18 Nov., at 23:00, to be delivered in Devilry

Goals

- Learn to use *Gensim* library to deal with word embeddings in *Python* (or any other toolkit on your choice).
- Learn to evaluate existing distributional models.
- Employ pre-trained word embedding models to build a document classifier.

Introduction

Modern distributional semantic algorithms (also known as **word embedding** algorithms) can be found inside many intelligent systems dealing with natural language. They became especially popular after the introduction of prediction-based models based on artifical neural networks, like **Continuous Bag-of-Words** and **Continuous Skip-gram** algorithms, first implemented in *word2vec* tool. Their ultimate aim is to learn meaningful vectors (embeddings) for words in natural language, such that semantically similar words have mathematically similar vectors.

It can be profitable to try to implement a distributional algorithm from scratch (we encourage you to do this); however, in real life tasks it is usually much more convenient to employ already available implementations and train your own models with them, or even re-use some of the pre-trained models in your specific problems. Thus, the current assignment aims to make you familiar with some of this software. We suggest *Gensim* library for *Python*, but you are free to use other existing tools: *TensorFlow*[1], original *word2vec*[2], *deeplearning4j*[3], etc. We provide some pre-code for *Python* and *Gensim*.

Please make sure you read through the entire assignment before you start. If you have any questions, please email andreku@ifi.uio.no, and make sure to take advantage of the group sessions. Solutions must be submitted through Devilry[4]

[1] `https://www.tensorflow.org/tutorials/word2vec/`
[2] `https://code.google.com/archive/p/word2vec/`
[3] `https://deeplearning4j.org/word2vec`
[4] `https://devilry.ifi.uio.no/`

by 23:00 on November 18. Please upload a single .tgz archive including a PDF with your answers, (heavily commented) code and data files.

Recommended reading

1. **Speech and Language Processing**. Daniel Jurafsky and James Martin. 3rd edition draft of April 9, 2016. Chapter 15, 'Vector Semantics'[5].
2. **Distributed representations of words and phrases and their compositionality**. Mikolov, Tomas, et al., 2013.[6]
3. **Speech and Language Processing**. Daniel Jurafsky and James Martin. 3rd edition draft of April 11, 2016. Chapter 16 'Semantics with dense vectors'[7].
4. `http://radimrehurek.com/gensim/models/word2vec.html`
5. `https://rare-technologies.com/word2vec-tutorial/`

1 Basic operations

1.1 Semantic Vectors web service

Make yourself familiar with the *Semantic Vectors* web service (`http://ltr.uio.no/semvec`). To play with it, you don't need to install or download anything. *Semantic Vectors* features pre-trained word embedding models for **English** (trained on the British National Corpus, Wikipedia and Google News) and **Norwegian** (trained on Norsk Aviskorpus and NoWAC). All the models were trained on lemmatized corpora using the **Continuous Skip-gram** algorithm. You can produce nearest semantic associates for any given word, calculate cosine similarity between pairs of words, visualize words' location in the reduced 2-dimensional space, etc. Briefly describe in your report what features you feel are missing from the service.

1.2 Working with models locally

Web services are good for quickly demonstrating a technology, but for the majority of real-world tasks you would like to have data, models and code at your hands, especially if you have lots of data to process. We will now work locally with the models functioning under the hood of *Semantic Vectors*.

Download the pre-trained English models for **BNC** and **Wikipedia**, listed at `http://ltr.uio.no/semvec/en/about#models`. They are made available in the standard binary *wordvec* format (gzipped) and can be loaded into nearly any piece of software able to work with predictive distributional models[8]. If you use *Python* (recommended), we provide a sample script at `http://ltr.uio.`

[5] `https://web.stanford.edu/~jurafsky/slp3/15.pdf`

[6] `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionalit` `pdf`

[7] `https://web.stanford.edu/~jurafsky/slp3/16.pdf`

[8] You can use the *Gensim* method *save_word2vec_format(binary=False)* to save these models in text format, making them easier for visual inspection.

`no/~andreku/5820/play_with_model.py`. In order to use it, you should first install *Gensim*[9];

Play with the downloaded models by running the script with the model filename as an argument:

```
python play_with_model.py enwiki.bin.gz
```

It will load the chosen model and invite you to enter a query word. If the word is present in the loaded model, its 10 nearest associates together with their cosine similarity to the query word will be printed. If you enter several words separated by spaces, the model will try to find out which of them is the most semantically distant from the others ('doesn't belong here'): for example, '*fire*' doesn't belong to the list '*orange, apple, pineapple*'.

Find out which *Gensim* methods the code uses (you can consult the manual from the **Recommended reading** section). You should be most interested in the *most_similar* method. Your task is to analyze the content words from the first sentence of the abstract to the paper '*Distributed Representations of Words and Phrases and their Compositionality*' by Mikolov et al. (see the **Recommended reading** section). Particularly, you are to use the **English Wikipedia** model to find the not-so-nearest semantic associates for these words: not the first 10 associates (those can be easily produced from the web service we mentioned above), but **the next 5**.

1. Modify the provided script (or write your own) so that it was able to take a text file with query words (one word per line) as an input;
2. create such an input file with all the (lemmatized and lower-cased) content words from the first sentence of the abstract;
3. modify the script so that for each input word it outputs its **11th, 12th, 13th, 14th and 15th nearest semantic associates** (with the cosine similarities);
4. report the produced associates and similarities.

2 Evaluating word embedding models

After training a word embedding model, one is usually interested in discovering how good it actually is. One of the popular methods to evaluate distributional models is to use the so called **analogy datasets**: quadruplets or proportions of semantically related words, in which a model has to guess the last element. For example, for the sequence 'Paris, France, Oslo, **???**' the model should output 'Norway', thus making an **analogical inference**. *Gensim* already features the *accuracy* method which implements this evaluation approach. In fact, the sample script we provided will evaluate the loaded model in this way if you supply the path to an analogy dataset file as a second argument:

```
python play_with_model.py enwiki.bin.gz questions-words.txt
```

1. download the analogy dataset introduced by Tomas Mikolov: `http://ltr.uio.no/~andreku/5820/questions-words.txt`;

[9] `http://radimrehurek.com/gensim/install.html`

2. visually inspect the file, make sure you understand its format;

3. evaluate the **BNC** and the **English Wikipedia** models using the downloaded dataset as the gold standard;

4. (optional) if your RAM size allows you to load the **Google News** model (also listed at `http://ltr.uio.no/semvec/en/about#models`), evaluate it as well;

5. report the total accuracy for each model.

Analogy datasets evaluate the embeddings' ability to capture complex semantic relations between 'constellations' of words. However, there is another (and in fact more traditional) approach to test distributional models: measuring how good they are in reproducing human experts' judgments about **semantic similarity between pairs of words**.

The idea is that we ask a significant amount of human annotators to rank word pairs according to their similarity and calculate cosine similarities for these pairs in the model under evaluation. Then we simply measure the Spearman rank-order correlation coefficient[10] between the two lists of similarities. Correlation close to **1** means that the model is extremely good in mimicking human judgments, and thus is supposedly superior in most downstream tasks. There are many published semantic similarity datasets, and lots of academic discussion goes on around them.

Unfortunately, there is no automated way to evaluate distributional models with similarity datasets in *Gensim* yet. Your task is to implement it.

1. Download **Simlex-999** dataset from `https://www.cl.cam.ac.uk/~fh295/simlex.html`;

2. read its description on the web page;

3. visually inspect the dataset, make sure you understand its format;

4. write code that takes as an input the dataset file and the model file and outputs the model's correlation with **Simlex-999** similarity judgments.

   - to produce cosine similarities for word pairs with *Gensim*, you would need the *similarity* method
   - Spearman correlation between lists of real values can be calculated in *Python* using *scipy.stats.spearmanr* function[11].

5. Report evaluation results for all the downloaded models.

6. Are the results different from the analogy dataset evaluation? If so, propose possible explanations.

[10] `https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient`

[11] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html`

3 Document classification with word embeddings

Distributional semantics can represent entities larger than words: phrases, sentences and whole documents. This makes it possible to train semantically-aware text classifiers. In this task, you will have to build document representations based on the same pre-trained English models that we played with above.

Our texts come from the Signal Media dataset[12], which contains one million English news texts from September 2015. Download the archive here:`http://ltr.uio.no/~andreku/5820/signal_lemmatized.tar.gz`. This is a sample from the whole dataset, with several thousand of news articles from 4 sources:

- **Individual.com** (general news aggregator);
- **4Traders** (trading, stock exchange news);
- **Mail Online UK** (British newspaper *The Daily Mail*);
- **Latest Nigerian News** (news from Nigeria).

Each file corresponds to one source, inside the files one line corresponds to one news article. Data format (tab-separated):

- Date
- Source (**class**)
- Type (always *News* in this sample)
- Text

The texts are already lemmatized, and stop words are removed, thus, they are ready for processing with distributional models.

Suppose we want to train a classifier which will be able to predict the source of a news text based on the words used in it. For this, we need to somehow jump from *documents as character strings* to *documents as numerical representations* that can be fed to any mainstream machine learning algorithm.

Of course, one obvious way to do this is to represent all the documents as **bags-of-words**: that is, extract the collection vocabulary (the union of all word types in the texts) and count frequency of each word type in each document. Documents are then represented as sparse vectors of the size equal to the size of the vocabulary. Machine learning classifiers can be easily trained on such data.

However, we would like to:

1. **leverage semantic information** contained in the pre-trained distributional models;
2. preferably work with **dense low-dimensional vectors**, not with sparse high-dimensional vectors produced by the **bag-of-words** approach.

Your task is to come up with semantically-aware representations of the documents in this collection and test how good they are in predicting documents' classes.

1. Write code to extract texts from the dataset;

[12] `http://research.signalmedia.co/newsir16/signal-dataset.html`

2. Implement any approach that uses pre-trained distributional models to produce meaningful dense representations of the extracted texts:
    - the most straightforward way to do this is with the **semantic fingerprints** algorithm (averaging embeddings of words in the document), which will be extensively covered in the Lecture 4 'Beyond words: distributional representations of texts';
    - however, you are free to use other approaches: **Paragraph Vector** (doc2vec)[13], **deep inverse regression**[14], or anything else.
3. convert lemmatized texts to dense vector representations and save them into one file, in which each line corresponds to one news text, starts with the class label followed by a tab, and continues with the tab-separated components of the dense vector corresponding to this text (see a toy example at `http://ltr.uio.no/~andreku/5820/example_dataset.tsv`);
4. use the provided *Python* script at `http://ltr.uio.no/~andreku/5820/classifier.py` to read your dataset, train a standard logistic regression classifier and evaluate it (the script requires *scikit-learn* library installed[15]):
    - `python classifier.py dataset.tsv`
5. the script will output precision, recall and f1-score both on the raw training set and with the 10-fold cross-validation;
6. report these values for all the pre-trained models you used.

Compare the performance of different models. What model provided the best embeddings for this classification task? Why is that so, in your opinion?

Happy coding!

[13] `https://github.com/RaRe-Technologies/movie-plots-by-genre/blob/master/Document%20classification%20with%20word%20embeddings%20tutorial.ipynb`

[14] `https://github.com/TaddyLab/gensim/blob/deepir/docs/notebooks/deepir.ipynb`

[15] `http://scikit-learn.org/stable/install.html`