Learning the notion of similarity

Aprendiendo la noción de similaridad

David Gordo
Investigador Postdoctoral
ICMAT CSIC

11 de diciembre de 2019

Outline

Generalities

- ◇ Project in collaboration with a private company that provides **legal information services** to lawyers
- ◇ Goal: Create **smarter** products incorporating AI
- ◇ Large dataset of **unlabeled** legal sentences ($\sim$ 0.5 M)
- ◇ Small dataset of **labeled** legal sentences for classification tasks

We developed multiple tools involving NLP in legal texts:

- Verdict analysis
- Named-entity recognition (NER)
- Classification of documents
- **Information retrieval system**

One specific application: Information retrieval system

- Given a reference text, we want to find the legal texts in our corpus that are *similar*. Ranking problem
- The key goal is to define a **notion of similarity between documents**
- Technical constraint: no a priori notion of similarity given by the experts, **subjective** idea
- Our solution: latent vectorial space in which

$$\text{close} = \text{similar}$$

Subtleties

- Really **specific context**: legal text + Spanish language
- Pre-trained word/document representations give bad results, we need to train them using our legal corpus
- **Labeling is expensive** for all tasks, you need experts
- You cannot design a labeling experiment without a previous model (low a priori probability)

We need to build document representations from scratch, using an unsupervised or semi-supervised method

Representing words: word embeddings

One-hot encoding

- Sparse representation
- No semantic information
- No distance notion
- Computationally inefficient

Rome Paris word V

```
Rome   = [1, 0, 0, 0, 0, 0, ..., 0]
Paris  = [0, 1, 0, 0, 0, 0, ..., 0]
Italy  = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]
```

$|V| \simeq 40.000$

Word embeddings

- Dense representation
- Captures semantic information
- Vector algebra manipulations
- Computationally efficient

$\texttt{Rome}: [0.364, -0.216, 0.035, \ldots, 0.115]$

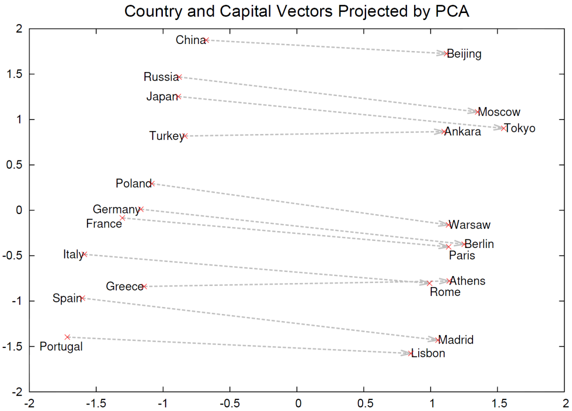
$\texttt{Paris}: [0.152, -0.117, 0.024, \ldots, 0.218]$

$\texttt{Italy}: [0.451, 0.219, -0.024, \ldots, 0.351]$

$\texttt{France}: [0.115, 0.178, -0.504, \ldots, 0.332]$

$d \simeq 300$

Word analogies



$$w_{\text{king}} - w_{\text{man}} + w_{\text{woman}} \simeq w_{\text{queen}}$$
$$w_{\text{paris}} - w_{\text{france}} + w_{\text{italy}} \simeq w_{\text{rome}}$$

$$w_{\text{einstein}} - w_{\text{scientist}} + w_{\text{painter}} \simeq w_{\text{picasso}}$$
$$w_{\text{his}} - w_{\text{he}} + w_{\text{she}} \simeq w_{\text{her}}$$

Unsupervised document embedding

Similar ideas can be applied to create fixed-size document embeddings in an unsupervised way:

- doc2vec: adds a vector representation of the document to the word2vec algorithm
  Prohibitive for production

  [Le and Mikolov (2014)]

- RNN based models: training to reconstruct surrounding sentences in a text (cannot be applied to our case)

  [Kiros et al (2015)]

- Average over the word embeddings of the text
  No order, all words are equally important
- **Term frequency-inverse document frequency** (tf-idf) weighted average over the word embeddings of the text
  No order, fast in prediction

Strategy

In order to attack the problem, we applied a

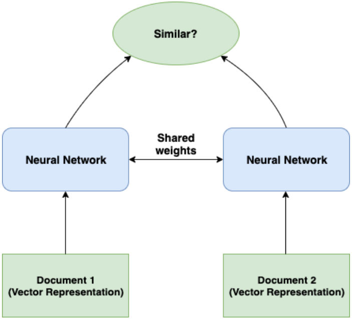
semi-supervised approach

I. Create baseline model of similarity: **tf-idf weighted average** of word embedding with cosine similarity
   We can *recycle* the word embeddings used for other tasks
II. Perform an experiment to **label** some of the sentences and **check** the validity of the baseline model
   Proposals acceptance ratio is significant
III. **Refine** the document embedding using labeled data (small dataset)

Refinement

Subrogate classification problem to accommodate **experts knowledge**

Siamese network architecture to train NN refinying the embedding



- Distance function measuring the similarity (Loss)
- Neural Network refining the embedding
- Input data: Baseline document embeddings

Differenciability needed to train through stochastic gradient descent. Easily implemented in Automatic Differentiation library (tensorflow).

Refinement: NN

- The siamese architecture is built to transform our refinement problem into a classification one, where a loss function can be defined
- All learnable parameters are in the NN, which will transform each document embedding into a refined one
- We can use this step to reduce the dimension of the embedding, this will help when looking for similar documents in production (large corpus)
- The architecture of the NN depends on the amount of labeled data. We used a linear transformation, since deeper networks quickly overfit

Results

ref	candidate	similar	cos	cos_new
7e148604	7df0c109	True	0.996387	0.969654
7e148604	7da39504	True	0.995567	0.975855
7e148604	7df01504	False	0.978922	-0.338358
7e148604	7da0fe03	False	0.984282	0.270300
7e148604	7df38f05	False	0.996457	0.968259
7e148604	7df3cc0c	False	0.970790	-0.115664

Ranking metrics:

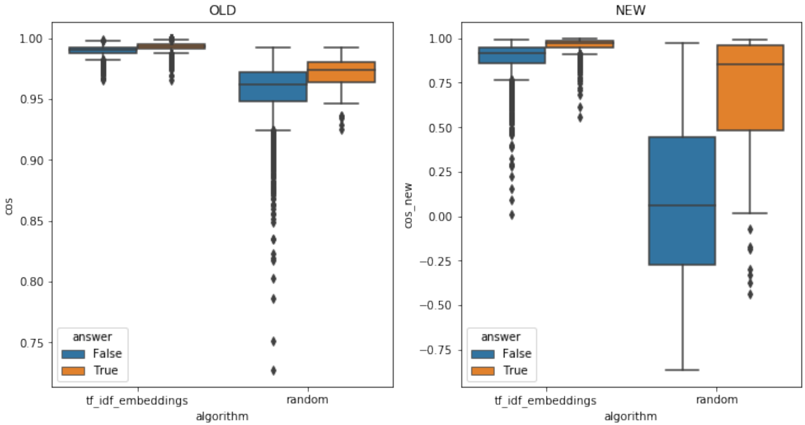
- Percentage of sentences *correctly ranked*:

  Old 62%. New 69%.

- Average gap between smallest similar and largest non-similar cosine similarity:

  Old 0.009. New 0.111.

Results



The cosine similarity is now better discriminating if two sentences are similar or not.

Conclusions

- We proposed a method to compute document similarity in a highly specific context
- Can be applied without labeled data, and refined in an iterative manner when data is collected
- Plenty of freedom for the NN
- General idea:
  can be applied to problems with different kind of data

Future work:

- Repeat the experiment and explore more complex NN
- Interpretability, robustness.
  This tools will assist in important decision making

Gracias

www.PlanTL.es

PlanTecnologiasLenguaje@mineco.es

BACKUP slides

Language model

- Determine the probability of a sentence s in a given domain

$$P(s = w_1 \cdots w_n) = \prod_{k=1}^{n} P(w_k | w_1 \cdots w_{k-1})$$

- In order to train a language model, you *just* need plain text. Unsupervised
- We can use the creation of a language model as a secondary problem to train an useful mathematical representation of words
- Distributional hypothesis: "You shall know a word by the company it keeps"

[John R. Firth (1957)]

Traditional approach: n-gram model

n-gram assumption (Markov): the probability of a word depends only on the previous $n-1$ words

Limitations:

- Curse of dimensionality: n-gram model on a corpus of vocabulary size V requires computing V^n probabilities.
- Usually $n \sim 10$, $V \sim 10K, 1M$. You need a huge amount of data to train it. More data $\rightarrow$ larger V
- Word similarity ignorance: the *one-hot* representation of the words stores no information about their meaning
- Sparse representation: computationally inefficient

Rome Paris word V

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
```

Text representation: BoW

Common fixed-size vector representation of text:
Bag of Words (BoW) or Bag of *n*-grams

[Harris (1954)]

- Word order is lost, it only counts the occurrences of each word in the text
- Bag of *n*-grams considers short-order context, but suffers from sparsity and high dimensionality
- No sense about semantics or distance between words

$$x = (0, \ldots, \overset{\text{italy}}{0}, \overset{\text{dog}}{1}, 0, \ldots, 0, \overset{\text{jumped}}{1}, 0 \ldots)$$

BACKUP slides

Distributed representations

Distributed representations or Word embeddings are word representations obtained from neural network based language models.

[Bengio (2001)]

- We have a mapping transform each $w_i \in V \rightarrow \mathbb{R}^d$
- Each word is represented as a dense vector.
- They encode a similarity concept (topology)
- Computationally efficient
- Main advantage: Generalization power

dog = (0 , … , 1 , …, 0)

$\longleftrightarrow$ V

dog = (0.12 , … , -0.32)

$\longleftrightarrow$ d << V

Representing words: word embeddings

A frequently used word embedding algorithm is word2vec

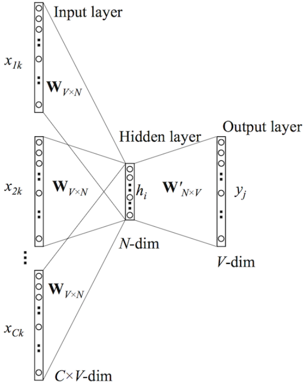
[Mikolov et al. (2013)]

- Simple network (linear transformation) to improve efficiency
- Faster training allows to use larger datasets
- Quality of word representations improves significantly with more training data
- Context both from previous and next words (window)
- Similarity metric given by cosine similarity
- The resulting representations contain surprisingly a lot of syntactic and semantic information (word analogies)
- Other proposals: Glove, FastText (character n-grams), ...

word2vec: Training

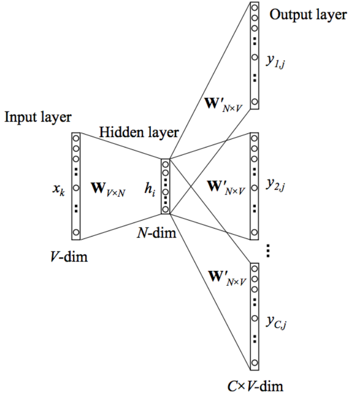
Two algorithms for learning word vectors:

Continuous Bag of Words (CBOW)



Predict word given its context

Skip-gram



Predict context given a word

BACKUP slides

Results