Gidi Shperber Follow
Machine learning addict
Jul 26, 2017 · 8 min read

A gentle introduction to Doc2Vec

TL;DR

In this post you will learn what is **doc2vec**, how it's built, how it's related to **word2vec**, what can you do with it, hopefully with no mathematic formulas.

Intro

Numeric representation of text documents is a challenging task in machine learning. Such a representation may be used for many purposes, for example: document retrieval, web search, spam filtering, topic modeling etc.

However, there are not many good techniques to do this. many tasks use the well known but simplistic method of **bag of words (BOW)**, but outcomes will be mostly mediocre, since BOW loses many subtleties of a possible good representation, e.g consideration of word ordering.

Latent Dirichlet Allocation (LDA) is also a common technique for topic modeling (extracting topics/keywords out of texts) but it's very hard to tune, and results are hard to evaluate.

In this post, I will review the **doc2vec** method, a concept that was presented at 2014 by **Mikilov and Le** in this article, which we are going to mention many time through this post. Worth to mention that Mikilov is one of the authors of word2vec as well.

Doc2vec is a very nice technique. It's easy to use, gives good results, and as you can understand from it's name, heavily based on word2vec. so we'll start with a short introduction about word2vec.

word2vec

word2vec is a well known concept, used to generate representation vectors out of words.

There are many good tutorials online about word2vec, like this one and this one, but describing **doc2vec** without **word2vec** will miss the point, so I'll be brief.

In general, when you like to build some model using words, simply labeling/one-hot encoding them is a plausible way to go. However, when using such encoding, the words lose their meaning. e.g, if we encode *Paris* as *id_4*, *France* as *id_6* and *power* as *id_8*, *France* will have the same relation to *power* as with *Paris*. We would prefer a representation in which *France* and *Paris* will be closer than *France* and *power*.

The **word2vec,** presented in 2013 in this article, intends to give you just that: a numeric representation for each word, that will be able to capture such relations as above. this is part of a wider concept in machine learning—the feature vectors.

Such representations, encapsulate different relations between words, like synonyms, antonyms, or analogies, such as this one:

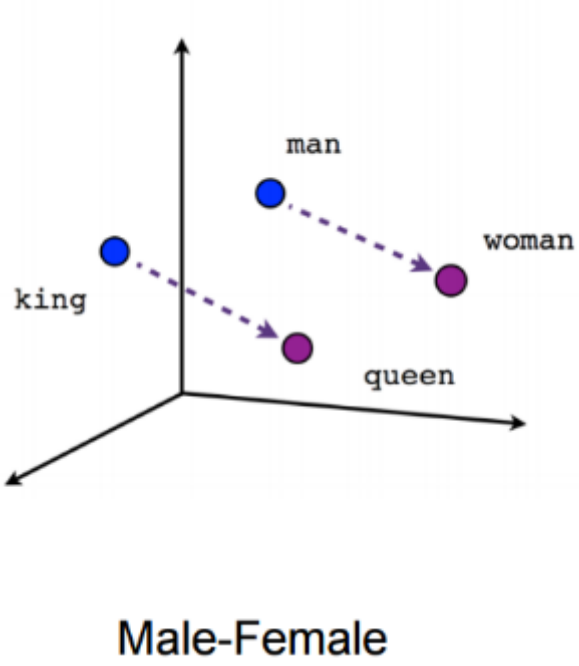


fig 1: king to queen is like man to woman. it is illegal to write about **word2vec** without attaching this plot

Word2vec algorithms

So how is it done? word2vec representation is created using 2 algorithms: Continuous Bag-of-Words model (**CBOW**) and the **Skip-Gram** model.

Continuous bag of words

continuous bag of words creates a sliding window around current word, to predict it from "context"—the surrounding words. Each word is represented as a feature vector. after training, these vectors become the word vectors.

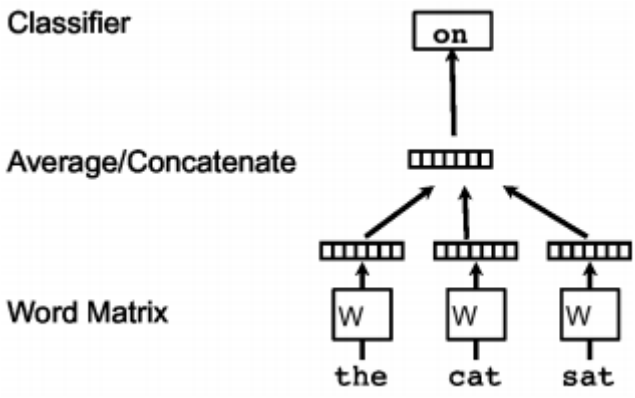


fig 2: CBOW algorithm sketch: the words "the" "cat" "sat" are used to predict the word "on"

As said before, vectors which represent similar words are close by different distance metrics, and additioanly encapsualte numeric relations, such as the *king-queen=man* from above.

Skip gram

The second algorithm (described in the same article, and well explaiend here) is actaully the opposite of CBOW: instead of prediciting one word each time, we use 1 word to predict all surrounding words ("context") **skip gram** is much slower than CBOW, but considered more accurate with infrequent words.

Doc2vec

After hopefully understanding what is **word2vec**, it will be easier to understand how **doc2vec** works.

As said, the goal of **doc2vec** is to create a numeric representation of a document, regardless of it's length. But unlike words, documents do not come in logical structures such as words, so the another method has to be found.

The concept that Mikilov and Le have used was simple, yet clever: they have used to **word2vec** model, and added another vector (Paragraph ID below), like so:

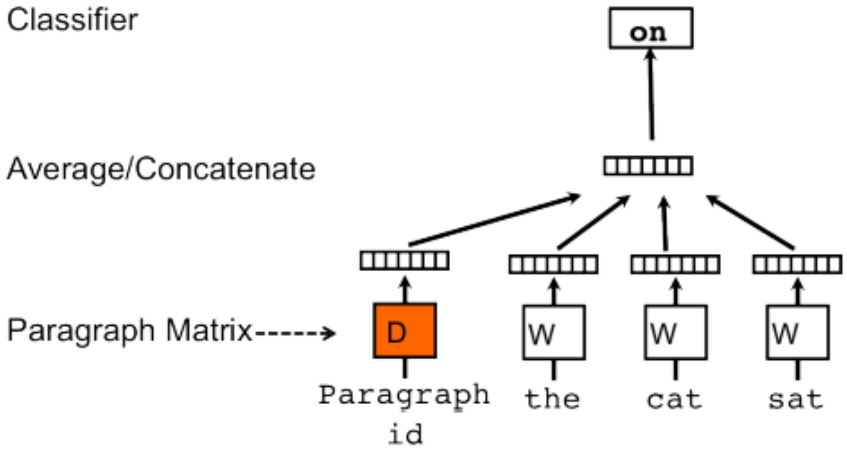


fig 3: PV-DM model

If you feel familiar with the sketch above, it's because it is a small extension to the CBOW model. but instead of using just words to predict the next word, we also added another feature vector, which is document-unique.

So, when training the word vectors W, the document vector D is trained as well, and in the end of training, it holds a numeric representation of the document.

The model above is called *Distributed Memory version of Paragraph Vector* (PV-DM). It acts as a memory that remembers what is missing from the current context—or as the topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document.

as in word2vec, another algorithm, which is similar to skip-gram may be used Distributed Bag of **Words version of Paragraph Vector** (PV-DBOW)

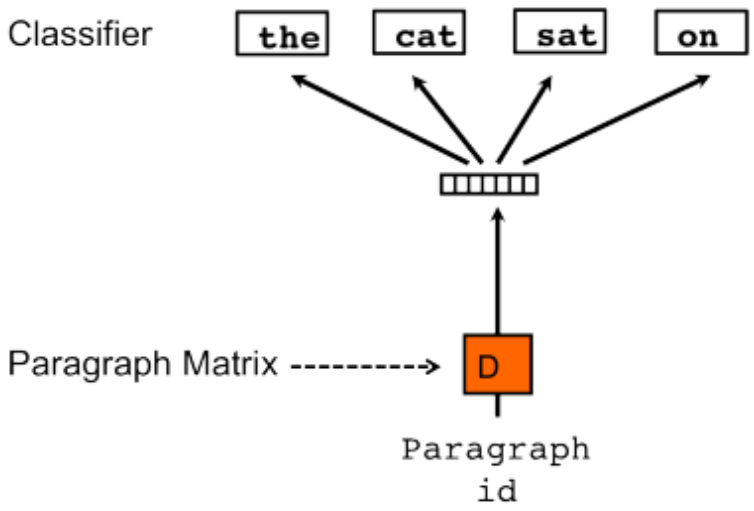


fig 4: PV-DBOW model

Here, this algorithm is actually faster (as opposed to **word2vec**) and consumes less memory, since there is no need to save the word vectors.

In the article, the authors state that they recommend using a combination of both algorithms, though the **PV-DM** model is superior and usually will achieve state of the art results by itself.

The **doc2vec** models may be used in the following way: for training, a set of documents is required. a word vector **W** is generated for each word, and a document vector **D** is generated for each document. The model also trains weights for a softmax hidden layer. In the inference stage, a new document may be presented, and all weights are fixed to calculate the document vector.

Evaluating model and some thoughts

The thing with this kind of unsupervised models, is that they are not trained to do the task they are intended for. E.g, **word2vec** is trained to complete surrounding words in corpus, but is used to estimate similarity or relations between words. As such, measuring the performance of these algorithms may be challenging. We already saw the *king ,queen,man, woman* example, but we want to make form it a rigorous way to evaluate machine learning models.

Therefore, when training these algorithms, we should be minded to relevant metrics. One possible metric for **word2vec**, is a generalization of the above example, and is called analogical reasoning. It contains many analogical combinations, here are some:

- *happy happily—furious furiously*

- *immediate immediately—infrequent infrequently*
- *slowing slowed—sleeping slept*
- *spending spent—striking struck*

A success in this task is getting very close results when caclulating distances between matching pairs.

dataset is availible at http://download.tensorflow.org/data/questions-words.txt.

Doc2vec was tested in the article on 2 tasks: the first is **sentiment analysis**, and the second one is similar to the analogical reasoning above.

Here are 3 paragraphs from the article. a dataset of such paragraphs was used to compare models. it is easy to see which 2 should be closer:

- **Paragraph 1:** calls from (000) 000 - 0000 . 3913 calls reported from this number . according to 4 reports the identity of this caller is american airlines .

- **Paragraph 2:** do you want to find out who called you from +1 000 - 000 - 0000 , +1 0000000000 or (000) 000 - 0000 ? see reports and share information you have about this caller
- **Paragraph 3:** allina health clinic patients for your convenience , you can pay your allina health clinic bill online . pay your clinic bill now , question and answers...

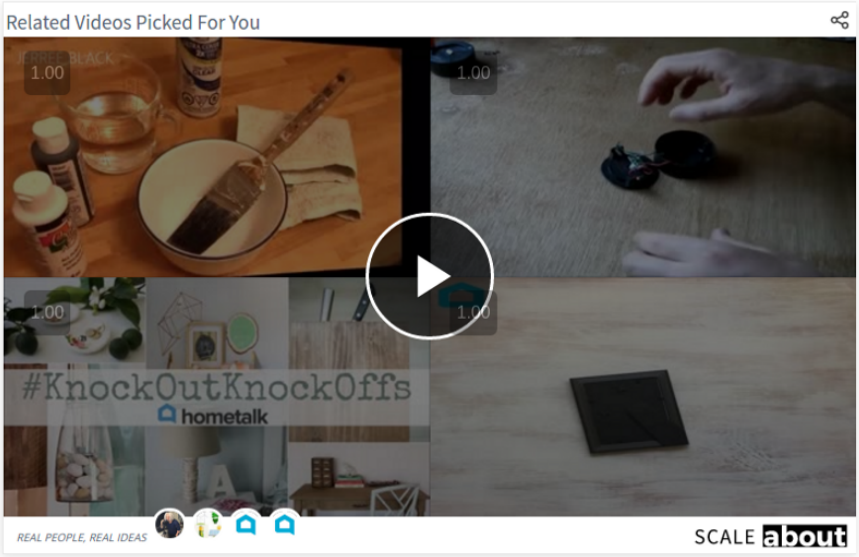
This dataset (was not shared as far as I know) was used to compare some models, and **doc2vec** came out as the best:

Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

Real life challenge—scale about

One of my clients, ***ScaleAbout***, uses machine learning methods to match "influencer" you-tube videos to content-articles. Doc2vec seems to be a great method for such match.

Here is an example for what the ***ScaleAbout*** does: in this article, about home made lights in a tree stump, you may see at the bottom 4 related video about woodworking stuff:



ScaleAbout's recommended videos for the above article

ScaleAbout current model uses tagging mechanism to tag the videos and the articles ("topic modeling") and measuring distance between tags.

ScaleAbout has a few corpora of text, related to the themes of its' clients. E.g, there is a 100K manually tagged documents about "do it yourself", for publishers such as above. There are 17 possible tags for each article (e.g, "home decor", "gardening", "remodeling and

renovating" etc.). For this experiment, we decided trying to predict the tags using doc2vec and some other models.

ScaleAbout's current best model was a convolutional neural network, on top of **word2vec**, which achieved accuracy of around **70%** in predicting the tags for the documents (as described here).

Doc2vec model by itself is an unsupervised method, so it should be tweaked a little bit to "participate" in this contest. Fortunately, as in most cases, we can use some tricks: If you recall, in *fig 3* we added another document vector, which was unique for each document. If you think about it, it is possible to add more vectors, which don't have to be unique: for example, if we have tags for our documents (as we actually have), we can add them, and get their representation as vectors.

Additionally, they don't have to be unique. This way, we can add to the unique document tag one of our 17 tags, and create a doc2vec representation for them as well! see below:

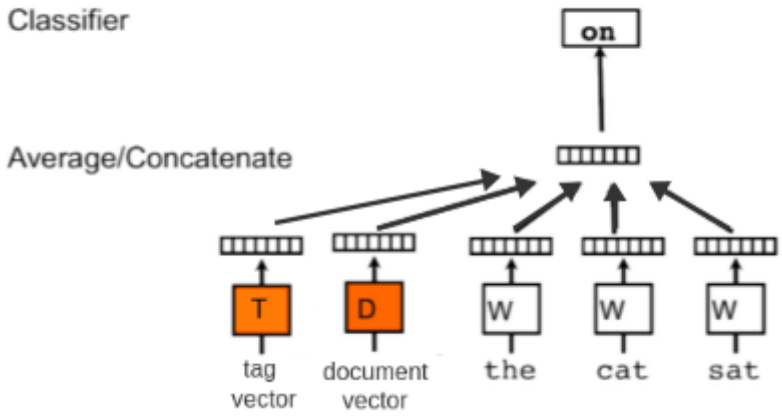


fig 5—doc2vec model with tag vector

we will use **gensim implementation** of **doc2vec.** here is how the gensim TaggedDocument object looks like:

```
In [168]: tagged_docs[3]

Out[168]: TaggedDocument(words=['aftershere', 'project', 'finishing', 'stages', 'home', 'decor', 'kitchen', 'design', 'beforeher
e', 'project', 'finishing', 'stages', 'home', 'decor', 'kitchen', 'design', 'afterhere', 'project', 'finishing', 'stag
es', 'home', 'decor', 'kitchen', 'design', 'beforehere', 'project', 'finishing', 'stages', 'home', 'decor', 'kitchen',
 'design'], tags=['Remodeling & Renovating', 'SENT_3'])
```

gensim TaggedDocument object. SENT_3 is the unique document id, remodeling and renovating is the tag

using **gensim** doc2vec is very straight-forward. as always, model should be initialized, trained for a few epochs:

```
In [134]: model = gensim.models.Doc2Vec(tagged_docs, dm = 0, alpha=0.025, size= 20, min_alpha=0.025, min_count=0)
          # 10K docs =15sec on SAinteg

In [163]: for epoch in range(10):
              if epoch % 2 == 0:
                  print ('Now training epoch %s'%epoch)
              model.train(tagged_docs_small,total_examples=model.corpus_count)
              model.alpha -= 0.002  # decrease the learning rate
              model.min_alpha = model.alpha  # fix the learning rate, no decay

Now training epoch 0
Now training epoch 2
Now training epoch 4
Now training epoch 6
Now training epoch 8
```

and then we can check the similarity of every unique **document** to every **tag**, this way:

```
In [172]: model.docvecs.similarity('Gardening & Landscaping','SENT_3')
Out[172]: 0.23512461864743456
```

The tag with highest similarity to document will be predicted.

Using this method, with training only on 10K out of our 100K articles, we have reached accuracy of 74%. better than before.

Summary

We have seen that with some tweaking, we can get much more from an already very useful **word2vec** model. This is great, because as said before, in my opinion representation of documents for tagging and matching has still a way to go.

Additionally, this show this is a great example for how machine learning models encapsulate much more abilities besides the specific task they were trained for. This can be seen in deep CNN's, which are trained for object classification, but can also be used for semantic segmentation or clustering images.

To conclude, if you have some document related task—this may be great model for you!

feel free to comment and correct. if there will be enough interest, I'll share the code.

Author: Gidi Shperber