Spotle.ai Study Material
Spotle.ai/Learn

Medical Reports Standardization

Document Clustering

Part 2

Intuition

The main assumption that Word2Vec (Mikolov, K. Chen, et al., 2013) relies upon is the following one: words with similar contexts have similar meaning. This is not a new idea, it was proposed back in 1954 by Harris. However, training a model on this premise has proven to be surprisingly effective. Word2Vec starts with a set of word vectors that are initialized randomly. It scans the corpus sequentially, always keeping a context window around each word it looks at. The algorithm computes the dot product between the target word and the context words and tries to minimize this metric. Each time two words are encountered in a similar context, their link, or spatial distance, is reinforced. The more evidence is found while scanning the corpus that two words are similar, the closer they will be. There is a last challenge to address. The basic model we just described only provides positive reinforcement towards making the vectors closer. With an infinite corpus, the minimum state would be that that all vectors would be in the same position, which is obviously not the desired effect. To address this situation, Word2Vec initially proposed a Hierarchical Softmax regulator. Later on, an alternative method called Negative Sampling was proposed. This last one is simpler and is more effective. The basic premise is that each time the distance between to vectors is minimized, a few random words are sampled and their distance to the target vector is

maximized. This way, it is ensured that nonsimilar words stay far from each other.

Consider the following similar sentences: Have a good day and Have a great day. They hardly have different meaning. If we construct an exhaustive vocabulary (let's call it V), it would have V = {Have, a, good, great, day}.

Now, let us create a one-hot encoded vector for each of these words in V. *In machine learning, one-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). A similar implementation in which all bits are '1' except one '0' is sometimes called one-cold. - Wiki*

Here, the length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.

$$have = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}, a = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}, good = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}, great = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}, day = \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}$$

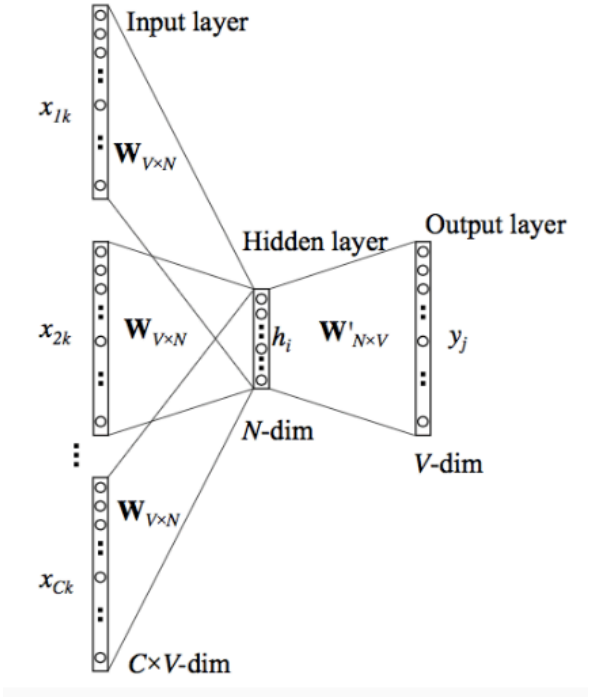
If we try to visualize these encodings, we can think of a 5-dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means 'good' and 'great' are as different as 'day' and 'have', or any two other words, which is not true.

Our objective is to create a vector for each word in lower dimensions (say 2 dimensions) so that 'good' and 'great' are closer than 'day' and 'have'.

CBOW Model

This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: Have a great day.

Let the input to the Neural Network be the word, great. Notice that here we are trying to predict a target word (day) using a single context input word great. More specifically, we use the one-hot encoding of the input word and measure the output error compared to one-hot encoding of the target word (day). In the process of predicting the target word, we learn the vector representation of the target word.



Training Data

PubMed probably the biggest database of open access articles in medicals. It specializes in the biomedical and pharmaceutical fields. The Open Access subset called PubMed Central contains 1.2M metadata records with more details than any other database. Most interestingly, the vast majority of those records contains high quality full-text bodies extracted from the original PDFs. PubMed has been identified as the best source for our current purposes. We have used around 1.1 million abstracts to build the model.

Python Library

A word embedding model is a model that can provide numerical vectors for a given word. Using the Gensim's downloader API, you can download pre-built word embedding models like Word2Vec, fastText, GloVe and ConceptNet. These are built on large corpuses of commonly occurring text data such as wikipedia, google news etc.

However, if you are working on a specialized niche such as technical documents or medical reports, you may not be able to get word embeddings for all the words. So, in such cases its desirable to train your own model.

Gensim's Word2Vec implementation lets you train your own word embedding model for a given corpus.

The following is a simple example that shows how easy it is to create our own model using Gensim.

```
from gensim.models.word2vec import Word2Vec
```

```
from nltk.tokenize import RegexpTokenizer

data = ["I love machine learning. Its awesome",
        "I love coding in python",
        "I love building chatbots",
        "they chat amagingly well"]

tokenizer = RegexpTokenizer(r'\w+')
```

```
# define training data
sentences = [tokenizer.tokenize(sent) for sent in data]

# train model
model = Word2Vec(sentences, size=2, min_count=1, seed=123456)


# access vector for one word
print('word vector for the word "love"')
print(model.wv['love'])

print('Cosine similarity between "chatbots" and "python" is {}'.format(
    model.wv.similarity('chatbots', 'python')))
```

```
word vector for the word "love"
[-0.1736126  -0.11730772]
Cosine similarity between "chatbots" and "python" is 0.3217918872833252
```

Training the model is fairly straightforward. You just instantiate Word2Vec and pass a of lists of words. Where each list within the main list contains a set of tokens from a user sentence. Word2Vec uses all these tokens to internally create a vocabulary that is a set of unique words.

Let's take a closer look at the parameter settings. To train the model earlier, we had to set some parameters. Now, let's try to understand what some of them mean.

size: The size of the dense vector to represent each token or word (i.e. the context or neighboring words). If you have limited data, then size should be a much smaller value since you would only have so many unique neighbors for a given word. If you have lots of data, it's good to experiment with various sizes. A value of 200 has worked well for this study.

window: The maximum distance between the target word and its neighboring word. If your neighbor's position is greater than the maximum window width to the left or the right, then, some neighbors would not be considered as being related to the target word. In theory, a smaller window should give you terms that are more related. We used 20 as window in this study.

min_count: Minimium frequency count of words. The model would ignore words that do not satisfy the min_count. Extremely infrequent words are usually unimportant, so it's best to get rid of those. We used 3 as min_count.

workers: Number of threads to use behind the scenes. We have a virtual server consisting of 75 cores and 124 GB of RAM. We assigned number of CPU that is 74 to worker.

iter: Number of iterations (epochs) over the corpus. We used a minimum of 20 iterations.

seed: Seed for random number generation.

Word similarity to document / sentence similarity

We have just seen how to find similarity between two words. It is easy since Word2Vec model gives vector of a word. We can use any similarity metric like cosine or Euclidean metric to compute similarity. But our objective is to find similarities between sentences.

Word Mover's Distance (WMD) is a method that allows us to assess the "distance" between two documents in a meaningful way, even when they have no words in common. It uses Word2Vec, vector embeddings of words. It been shown to outperform many of the state-of-the-art methods in k-nearest neighbours classification.

Even if the sentences have no words in common, but by matching the relevant words, WMD is able to accurately measure the (dis)similarity between the two sentences. The method also uses the bag-of-words representation of the documents. The intuition behind the method is that we find the minimum "traveling distance" between documents, in other words the most efficient way to "move" the distribution of document 1 to the distribution of document 2.

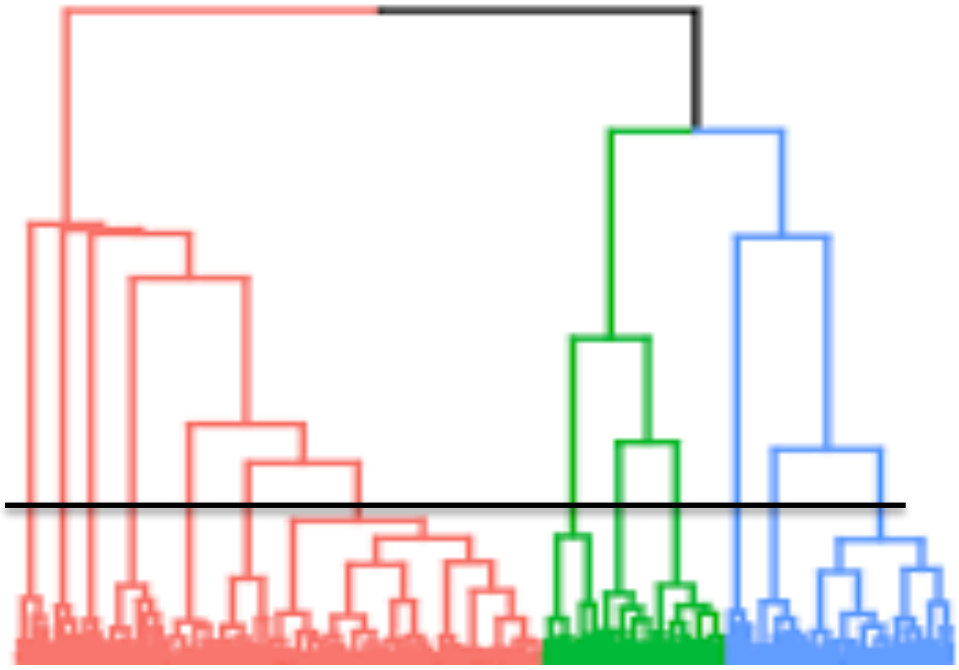
```
mode.wv.wmdistance(sentence1, sentence2)
```

'wmdistance' is function in Gensim to compute distance between two sentences or documents.

Document Clustering

Since we do not know number of clusters or groups at the beginning of sentence clustering, we use hierarchical clustering.

Later we cut the tree with at suitable height to get different clusters.



```
Z = cluster.hierarchy.linkage(X, method=method, metric=metric)
cutree = cluster.hierarchy.cut_tree(Z, height=cutoff)
```

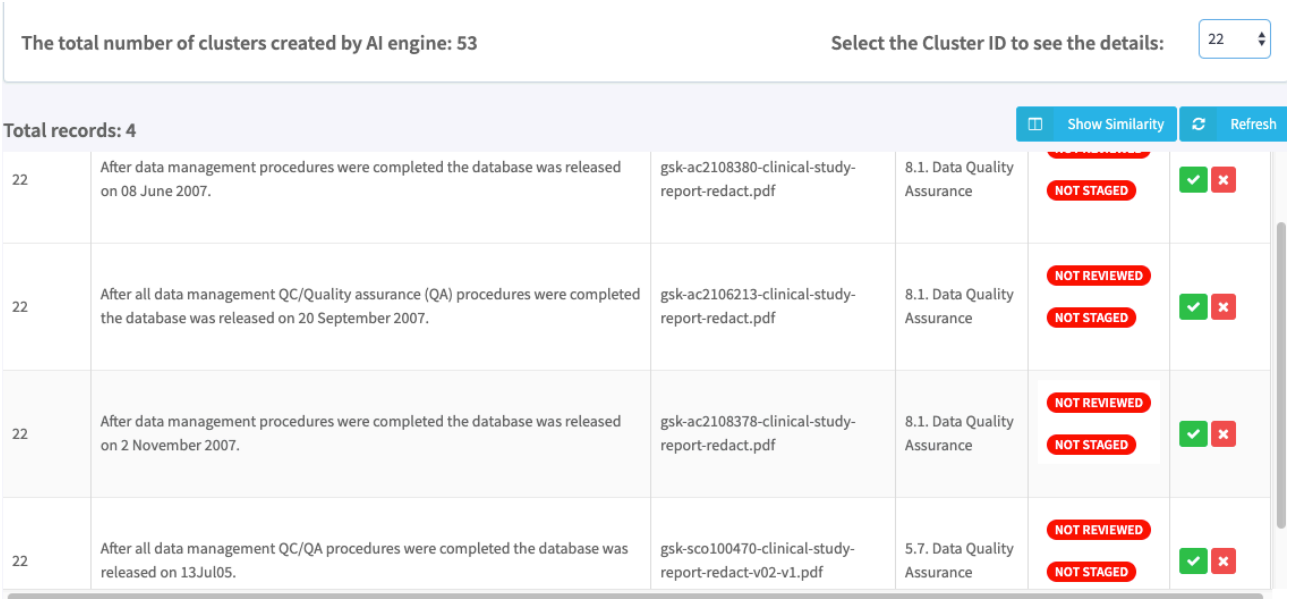
Outcome

'gsk-ac2106213-clinical-study-report-redact.pdf' is publicly available clinical study report (CSR). Following paragraph is taken from Data quality assurance section (section 8.1).

Subject data was recorded in paper CRFs at the clinical study site, by the investigator or their designee. A screening log with GSK required information, created by the investigator at site, tracked subjects who were screened prior to entry into the study but were

not randomised. Adverse events and concomitant medications were coded by the auto encoder using company standard dictionaries (GSK Drug) and industry standard dictionaries (MedDRA). After all data management QC/Quality assurance (QA) procedures were completed the database was released on 20 September 2007. The database was frozen on 5 October 2007 at which time access to the data was also restricted (database lock).

We have total 53 clusters formed from similarity matrix. That matrix is obtained using Word2Vec model and wmdistance function. The following shows 4 sentences from 4 different CSR which are very similar, and the clustering algorithm put them in a cluster.



All 4 sentences describe same context although they are different in text formation.
We are to choose one sentence that represent the cluster and will be included in the template.