Similarity Queries

Note

Click here to download the full example code

Demonstrates querying a corpus for similar documents.

```
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

Creating the Corpus

First, we need to create a corpus to work with. This step is the same as in the previous tutorial; if you completed it, feel free to skip to the next section.

```
from collections import defaultdict
from gensim import corpora

documents = [
    "Human machine interface for lab abc computer applications",
    "A survey of user opinion of computer system response time",
    "The EPS user interface management system",
    "System and human system engineering testing of EPS",
    "Relation of user perceived response time to error measurement",
    "The generation of random binary unordered trees",
    "The intersection graph of paths in trees",
    "Graph minors IV Widths of trees and well quasi ordering",
    "Graph minors A survey",
]

# remove common words and tokenize
stoplist = set('for a of the and to in'.split())
texts = [
    [word for word in document.lower().split() if word not in stoplist]
    for document in documents
]

# remove words that appear only once
frequency = defaultdict(int)
for text in texts:
    for token in text:
        frequency[token] += 1

texts = [
    [token for token in text if frequency[token] > 1]
    for text in texts
]

dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
```

Similarity interface

In the previous tutorials on Corpora and Vector Spaces and Topics and Transformations, we covered what it means to create a corpus in the Vector Space Model and how to transform it between different vector spaces. A common reason for such a charade is that we want to determine **similarity between pairs of documents**, or the **similarity between a specific document and a set of other documents** (such as a user query vs. indexed documents).

To show how this can be done in gensim, let us consider the same corpus as in the previous examples (which really originally comes from Deerwester et al.'s "Indexing by Latent Semantic Analysis" seminal 1990 article). To follow Deerwester's example, we first use this tiny corpus to define a 2-dimensional LSI space:

```
from gensim import models
lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=2)
```

For the purposes of this tutorial, there are only two things you need to know about LSI. First, it's just another transformation: it transforms vectors from one space to another. Second, the benefit of LSI is that enables identifying patterns and relationships between terms (in our case, words in a document) and topics. Our LSI space is two-dimensional (*num_topics* = 2) so there are two topics, but this is arbitrary. If you're interested, you can read more about LSI here: **Latent Semantic Indexing**:

Now suppose a user typed in the query *"Human computer interaction"*. We would like to sort our nine corpus documents in decreasing order of relevance to this query. Unlike modern search engines, here we only concentrate on a single aspect of possible similarities—on apparent semantic relatedness of their texts (words). No hyperlinks, no random-walk static ranks, just a semantic extension over the boolean keyword match:

```
doc = "Human computer interaction"
vec_bow = dictionary.doc2bow(doc.lower().split())
vec_lsi = lsi[vec_bow]  # convert the query to LSI space
print(vec_lsi)
```

Out:
```
[(0, 0.4618210045327158), (1, 0.07002766527900064)]
```

In addition, we will be considering **cosine similarity** to determine the similarity of two vectors. Cosine similarity is a standard measure in Vector Space Modeling, but wherever the vectors represent probability distributions, **different similarity measures** may be more appropriate.

Initializing query structures

To prepare for similarity queries, we need to enter all documents which we want to compare against subsequent queries. In our case, they are the same nine documents used for training LSI, converted to 2-D LSA space. But that's only incidental, we might also be indexing a different corpus altogether.

```
from gensim import similarities
index = similarities.MatrixSimilarity(lsi[corpus])  # transform corpus to LSI space and index it
```

Warning

The class `similarities.MatrixSimilarity` is only appropriate when the whole set of vectors fits into memory. For example, a corpus of one million documents would require 2GB of RAM in a 256-dimensional LSI space, when used with this class.

Without 2GB of free RAM, you would need to use the `similarities.Similarity` class. This class operates in fixed memory, by splitting the index across multiple files on disk, called shards. It uses `similarities.MatrixSimilarity` and `similarities.SparseMatrixSimilarity` internally, so it is still fast, although slightly more complex.

Index persistency is handled via the standard `save()` and `load()` functions:

```
index.save('/tmp/deerwester.index')
index = similarities.MatrixSimilarity.load('/tmp/deerwester.index')
```

This is true for all similarity indexing classes (`similarities.Similarity`, `similarities.MatrixSimilarity` and `similarities.SparseMatrixSimilarity`). Also in the following, *index* can be an object of any of these. When in doubt, use `similarities.Similarity`, as it is the most scalable version, and it also supports adding more documents to the index later.

Performing queries

To obtain similarities of our query document against the nine indexed documents:

```
sims = index[vec_lsi]  # perform a similarity query against the corpus
print(list(enumerate(sims)))  # print (document_number, document_similarity) 2-tuples
```

Out:
```
[(0, 0.998093), (1, 0.93748635), (2, 0.9984453), (3, 0.9865886), (4, 0.90755945), (5, -0.12416792), (6, -0.10639259),
```

Cosine measure returns similarities in the range *<-1, 1>* (the greater, the more similar), so that the first document has a score of 0.99809301 etc.

With some standard Python magic we sort these similarities into descending order, and obtain the final answer to the query *"Human computer interaction"*:

```
sims = sorted(enumerate(sims), key=lambda item: -item[1])
for i, s in enumerate(sims):
    print(s, documents[i])
```

Out:
```
(2, 0.9984453) Human machine interface for lab abc computer applications
(0, 0.998093) A survey of user opinion of computer system response time
(3, 0.9865886) The EPS user interface management system
(1, 0.93748635) System and human system engineering testing of EPS
(4, 0.90755945) Relation of user perceived response time to error measurement
(8, 0.050041765) The generation of random binary unordered trees
(7, -0.09879464) The intersection graph of paths in trees
(6, -0.10639259) Graph minors IV Widths of trees and well quasi ordering
(5, -0.12416792) Graph minors A survey
```

The thing to note here is that documents no. 2 (`"The EPS user interface management system"`) and 4 (`"Relation of user perceived response time to error measurement"`) would never be returned by a standard boolean fulltext search, because they do not share any common words with `"Human computer interaction"`. However, after applying LSI, we can observe that both of them received quite high similarity scores (no. 2 is actually the most similar!), which corresponds better to our intuition of them sharing a "computer-human" related topic with the query. In fact, this semantic generalization is the reason why we apply transformations and do topic modelling in the first place.

Where next?

Congratulations, you have finished the tutorials – now you know how gensim works :-) To delve into more details, you can browse through the **API Reference**, see the **Experiments on the English Wikipedia** or perhaps check out **Distributed Computing** in *gensim*.

Gensim is a fairly mature package that has been used successfully by many individuals and companies, both for rapid prototyping and in production. That doesn't mean it's perfect though:

- there are parts that could be implemented more efficiently (in C, for example), or make better use of parallelism (multiple machines cores)
- new algorithms are published all the time; help gensim keep up by **discussing them** and **contributing code**
- your **feedback is most welcome** and appreciated (and it's not just the code!): **bug reports** or **user stories and general questions**.

Gensim has no ambition to become an all-encompassing framework, across all NLP (or even Machine Learning) subfields. Its mission is to help NLP practitioners try out popular topic modelling algorithms on large datasets easily, and to facilitate prototyping of new algorithms for researchers.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('run_similarity_queries.png')
imgplot = plt.imshow(img)
plt.axis('off')
plt.show()
```

Out:

```
/Volumes/work/workspace/gensim_misha/docs/src/gallery/core/run_similarity_queries.py:194: UserWarning: Matplotlib is
  plt.show()
```

Total running time of the script: (0 minutes 0.663 seconds)

Estimated memory usage: 6 MB

Download Python source code: run_similarity_queries.py

Download Jupyter notebook: run_similarity_queries.ipynb

Gallery generated by Sphinx-Gallery