Gensim

March 14, 2016

1 Gensim

Gensim is a free Python library designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible.

Gensim aims at processing raw, unstructured digital texts (“plain text”). The algorithms in gensim, such as Latent Semantic Analysis, Latent Dirichlet Allocation or Random Projections, discover semantic structure of documents, by examining word statistical co-occurrence patterns within a corpus of training documents. These algorithms are unsupervised, which means no human input is necessary – you only need a corpus of plain text documents.

Once these statistical patterns are found, any plain text documents can be succinctly expressed in the new, semantic representation, and queried for topical similarity against other documents.

Library documentation: https://radimrehurek.com/gensim/index.html

In [1]: from gensim import corpora, models, similarities

   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"]

In [2]: # 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]

In [3]: # remove words that appear only once
   from collections import defaultdict
   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]

In [4]: from pprint import pprint
   pprint(texts)
In [5]: # create a dictionary mapping between ids and unique words
dictionary = corpora.Dictionary(texts)
print(dictionary)

Dictionary(12 unique tokens: ['computer', 'survey', 'minors', 'eps', 'time']...)

In [6]: # mapping between ids and words
print(dictionary.token2id)

{'computer': 0, 'survey': 7, 'minors': 11, 'eps': 8, 'time': 3, 'system': 4, 'interface': 1, 'human': 2, 'trees': 9, 'user': 6, 'response': 5, 'graph': 10}

In [7]: # convert the text to a bag-of-words corpus
corpus = [dictionary.doc2bow(text) for text in texts]
pprint(corpus)

[['(0, 1)', '(1, 1)', '(2, 1)'],
['(0, 1)', '(3, 1)', '(4, 1)', '(5, 1)', '(6, 1)', '(7, 1)'],
['(1, 1)', '(4, 1)', '(6, 1)', '(8, 1)'],
['(2, 1)', '(4, 2)', '(8, 1)'],
['(3, 1)', '(5, 1)', '(6, 1)'],
['(9, 1)'],
['(9, 1)', '(10, 1)'],
['(9, 1)', '(10, 1)', '(11, 1)'],
['(7, 1)', '(10, 1)', '(11, 1)']]

In [8]: # can convert to numpy/scipy matrices and back
from gensim import matutils
numpy_matrix = matutils.corpus2dense(corpus, num_terms=12)
print(numpy_matrix)

[[ 1.  1.  0.  0.  0.  0.  0.  0.  0.]
[ 1.  0.  1.  0.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  1.  0.  0.  0.  0.  0.]
[ 0.  1.  0.  0.  1.  0.  0.  0.  0.]
[ 0.  1.  1.  2.  0.  0.  0.  0.  0.]
[ 0.  1.  0.  0.  1.  0.  0.  0.  0.]
[ 0.  1.  1.  0.  1.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.  0.  0.  1.]
[ 0.  0.  1.  1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  1.  1.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  1.  1.  1.]
[ 0.  0.  0.  0.  0.  0.  0.  1.  1.]

In [9]: scipy_csc_matrix = matutils.corpus2csc(corpus)
numpy_corpus = matutils.Dense2Corpus(numpy_matrix)
scipy_corpus = matutils.Sparse2Corpus(scipy_csc_matrix)
In [10]: # initialize a TF-IDF transformation
tfidf = models.TfidfModel(corpus)

In [11]: # apply it to the whole corpus
corpus_tfidf = tfidf[corpus]
for doc in corpus_tfidf:
    print(doc)
[(0, 0.5773502691896257), (1, 0.5773502691896257), (2, 0.5773502691896257)]
[(0, 0.44424552527467476), (3, 0.44424552527467476), (4, 0.3244870206138555), (5, 0.44424552527467476),
 (1, 0.5710059809418182), (4, 0.4170757362022777), (6, 0.4170757362022777), (8, 0.5710059809418182)]
[(2, 0.49182558987264147), (4, 0.7184811607083769), (8, 0.49182558987264147)]
[(3, 0.6282580468670046), (5, 0.6282580468670046), (6, 0.45889394536615247)]
[(9, 1.0)]
[(9, 0.7071067811865475), (10, 0.7071067811865475)]
[(9, 0.5080429008916749), (10, 0.5080429008916749), (11, 0.6955464195200373)]
[(7, 0.6282580468670046), (10, 0.45889394536615247), (11, 0.6282580468670046)]

In [12]: # initialize an LSI transformation
lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=2)
lsi.print_topics(2)
Out[12]: ['0.703*"trees" + 0.538*"graph" + 0.402*"minors" + 0.187*"survey" + 0.061*"system" + 0.060*"time" + 0.060*"response" + 0.058*"user" + 0.049*"computer" + 0.035*"interface",
 -0.460*"system" + -0.373*"user" + -0.332*"eps" + -0.328*"interface" + -0.320*"response" + -0.320*"time" + -0.293*"computer" + -0.280*"human" + -0.171*"survey" + 0.161*"trees"]

In [13]: # create a double wrapper over the original corpus: bow->tfidf->lsi
corpus_lsi = lsi[corpus_tfidf]
for doc in corpus_lsi:
    print(doc)
[(0, 0.06600783396090387), (1, -0.5200703306361851)]
[(0, 0.19667592859142549), (1, -0.7609563167700046)]
[(0, 0.089926399724465117), (1, -0.72418606267525076)]
[(0, 0.07588476521782597), (1, -0.63205515860034256)]
[(0, 0.10150299184980183), (1, -0.57373084830029564)]
[(0, 0.70321089393783087), (1, 0.16115180214025826)]
[(0, 0.87747876731198271), (1, 0.1675890684659456)]
[(0, 0.90986246868185738), (1, 0.14066553628719076)]
[(0, 0.61658253505692762), (1, -0.05392907566393065)]

In [14]: # random projection model
rp = models.RpModel(corpus_tfidf, num_topics=500)

In [15]: # latent dirichlet allocation model
lda = models.LdaModel(corpus, id2word=dictionary, num_topics=100)
WARNING:gensim.models.lsimodel:too few updates, training might not converge; consider increasing the number of passes or iterations

In [16]: # convert a phrase into the LSI model space
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)
[(0, 0.07910475117449039), (1, -0.57328352430794061)]

In [17]: # index the transformed corpus from earlier
index = similarities.MatrixSimilarity(corpus_lsi)
In [18]: # perform a similarity query against the corpus using cosine similarity
   ...:     sims = index[vec_lsi]
   ...:     print(list(enumerate(sims)))
   ...:     
   ...:     [(0, 0.99994081), (1, 0.99330217), (2, 0.99990785), (3, 0.99984384), (4, 0.9992786), (5, -0.08804217), ...]

In [19]: # display in sorted order
   ...:     sims = sorted(enumerate(sims), key=lambda item: -item[1])
   ...:     pprint(sims)
   ...:     
   ...:     [(0, 0.99994081),
    ...:      (2, 0.99990785),
    ...:      (3, 0.99984384),
    ...:      (4, 0.9992786),
    ...:      (1, 0.99330217),
    ...:      (8, 0.22248439),
    ...:      (7, -0.016480923),
    ...:      (6, -0.0515742),
    ...:      (5, -0.08804217)]