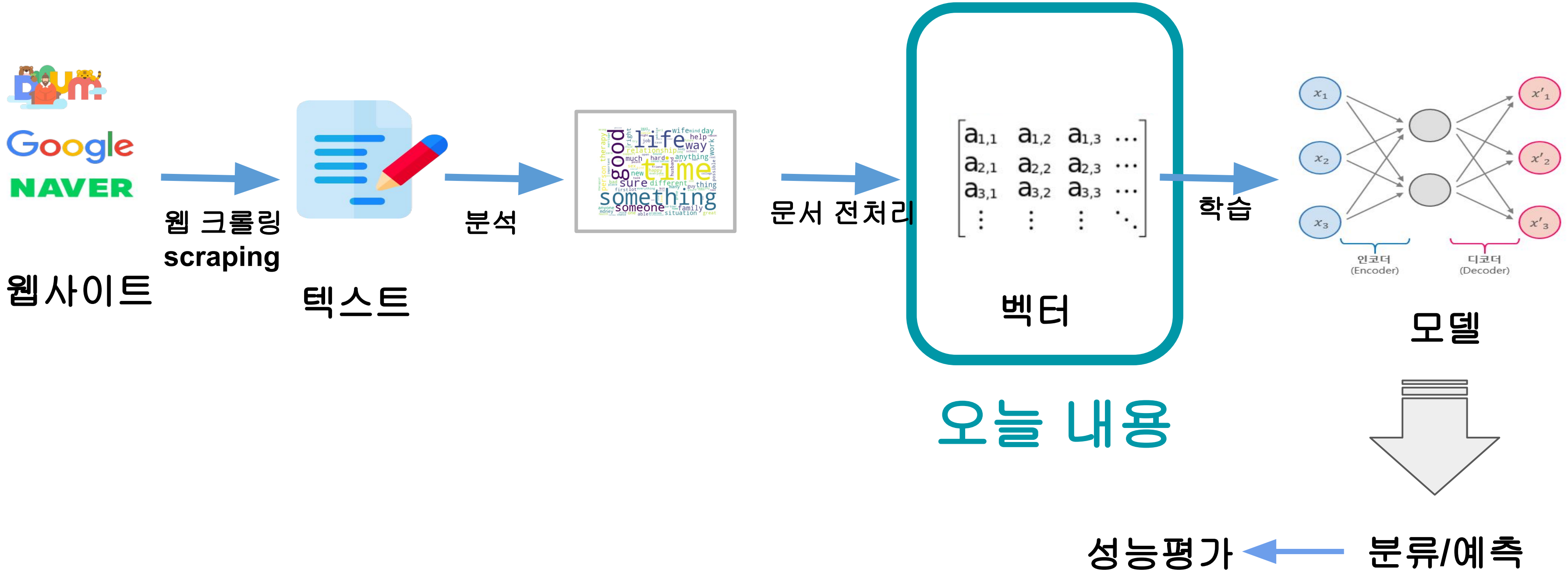
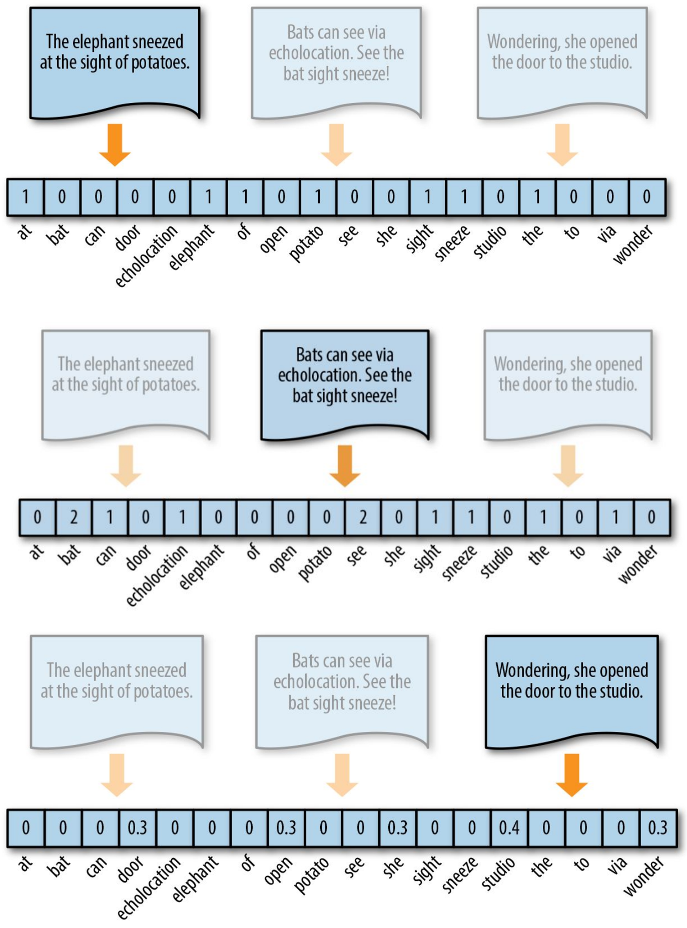
Topic Modeling

한눈에 보는 자연어 처리 과정




The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0
0 2 1 0 1 0 0 0 0 2 0 1 1 0 1 0 1 0
0 0 0 0.3 0 0 0 0.3 0 0 0.3 0 0 0.4 0 0 0 0.3
at bat can door echolocation elephant of open potato see she sight sneeze studio the to via wonder

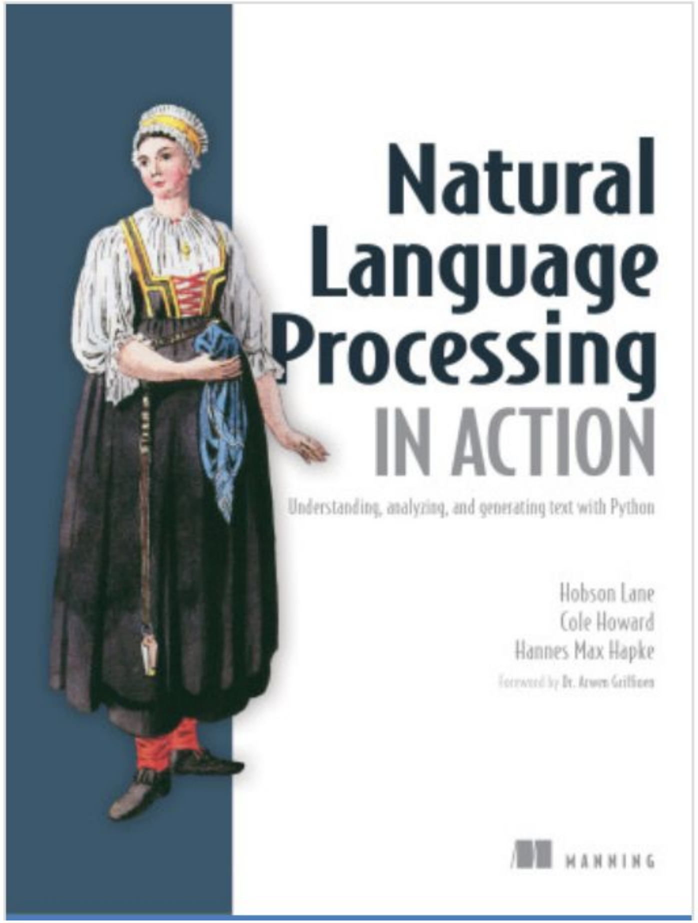
- ❑ TF-IDF vectors count the exact spellings of terms
- ❑ Synonyms with different spellings produce TF-IDF vectors that aren't close to each other in the vector space

Outline

- Analyzing semantics (meaning) to create topic vectors
- Semantic search using the similarity between topic vectors
- Scalable semantic analysis and semantic search for large corpora
- Using semantic components (topics) as features in your NLP pipeline
- Navigating high-dimensional vector spaces

Reference

- Natural Language Processing in Action
- Understanding, analyzing, and generating text with Python
- Hobson Lane, Cole Howard, Hannes Hapke



Natural Language Processing IN ACTION

Understanding, analyzing, and generating text with Python

Hobson Lane
Cole Howard
Hannes Max Hapke

Foreword by Dr. Arwen Griffioen

MANNING

Topic vectors

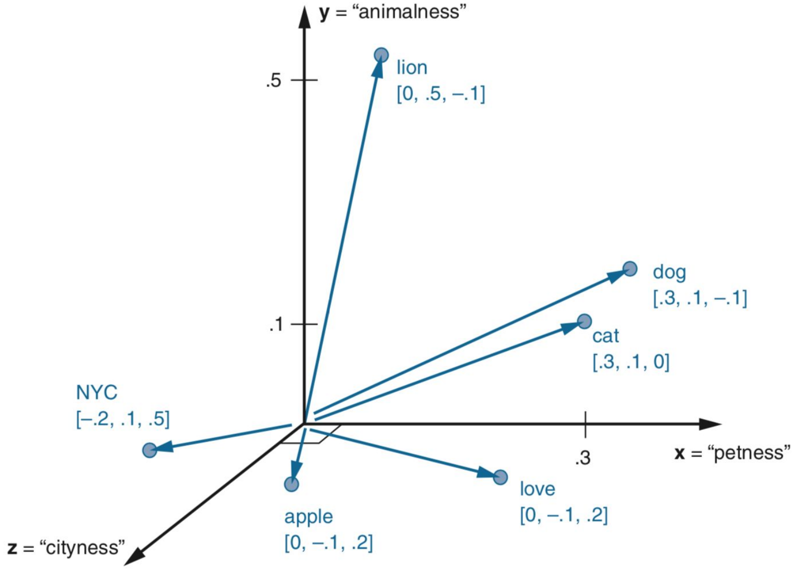
- Can represent the meaning of words and **the meaning of entire documents**
- We will use the BOW, TF-IDF vectors to compute the topic “scores” that make up the dimensions of topic vectors
- We will use the correlation of normalized term frequencies with each other to group words together in topics to define the dimensions of new topic vectors
- Topic vectors will help you do a lot of interesting things
  - search for documents based on their meaning—semantic search
  - identify the words (and n-grams) that best represent the subject (topic) of a statement, document, or corpus → can be further used for summarization
  - compare any two statements or documents and tell how “close” they are in meaning to each other

Thought experiment

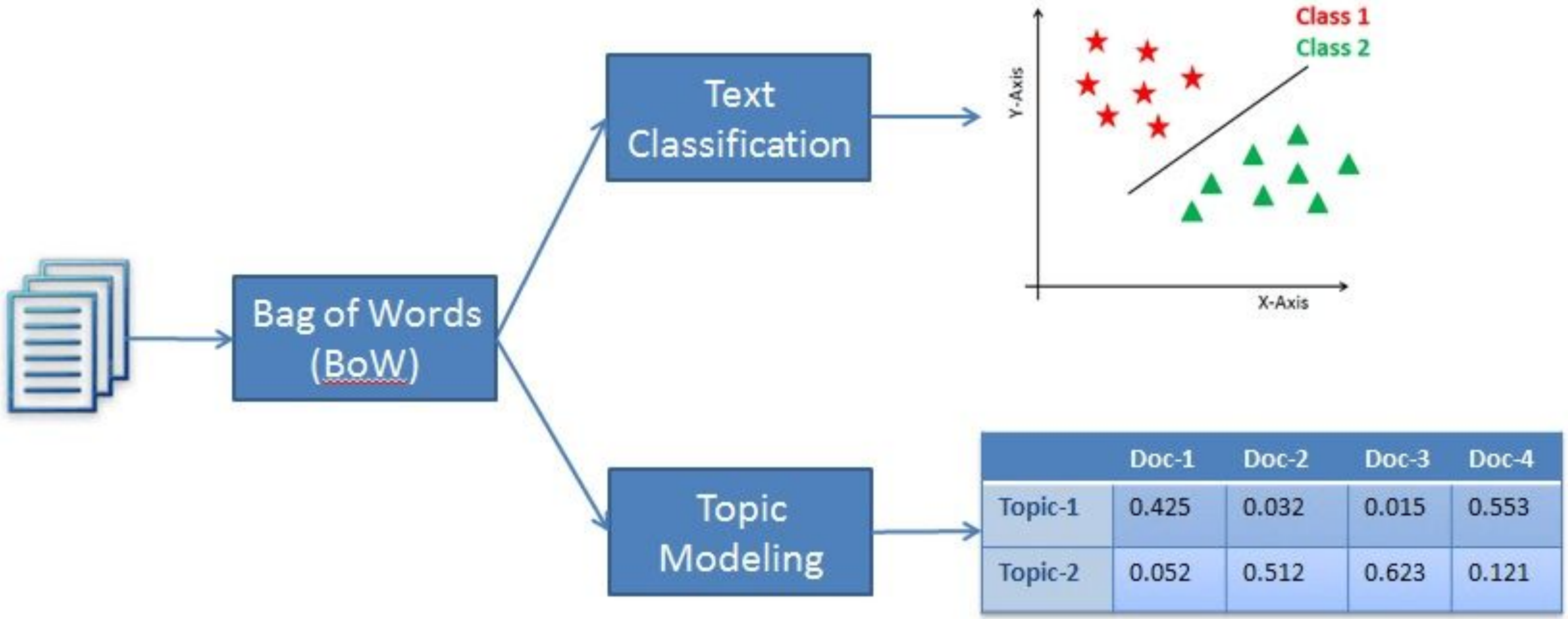
- Assume you have some TF-IDF vector for a particular document and you want to convert that to a topic vector -- how much each word contributes to your topics
- Create three topics: petness, animalness, and cityness
- “petness” topic will score words like “cat” and “dog” significantly, but probably ignore words like “NYC” and “apple.”
- If you “trained” your topic model like this, without using a computer, only your common sense, you might come up with some weights like this:

```
>>> topic = {}
>>> tfidf = dict(list(zip('cat dog apple lion NYC love'.split(),
...     np.random.rand(6))))
>>> topic['petness'] = (.3 * tfidf['cat'] +\
...                    .3 * tfidf['dog'] +\
...                     0 * tfidf['apple'] +\
...                     0 * tfidf['lion'] -\
...                    .2 * tfidf['NYC'] +\
...                    .2 * tfidf['love'])
>>> topic['animalness']  = (.1 * tfidf['cat']  +\
...                        .1 * tfidf['dog'] -\
...                        .1 * tfidf['apple'] +\
...                        .5 * tfidf['lion'] +\
...                        .1 * tfidf['NYC'] -\
...                        .1 * tfidf['love'])
>>> topic['cityness']    = ( 0 * tfidf['cat']  -\
...                        .1 * tfidf['dog'] +\
...                        .2 * tfidf['apple'] -\
...                        .1 * tfidf['lion'] +\
...                        .5 * tfidf['NYC'] +\
...                        .1 * tfidf['love'])
```

```
>>> word_vector = {}
>>> word_vector['cat']  =  .3*topic['petness'] +\
...                      .1*topic['animalness'] +\
...                       0*topic['cityness']
>>> word_vector['dog']  =  .3*topic['petness'] +\
...                      .1*topic['animalness'] -\
...                      .1*topic['cityness']
>>> word_vector['apple']=   0*topic['petness'] -\
...                      .1*topic['animalness'] +\
...                      .2*topic['cityness']
>>> word_vector['lion'] =   0*topic['petness'] +\
...                      .5*topic['animalness'] -\
...                      .1*topic['cityness']
>>> word_vector['NYC']  = -.2*topic['petness'] +\
...                      .1*topic['animalness'] +\
...                      .5*topic['cityness']
>>> word_vector['love'] =  .2*topic['petness'] -\
...                      .1*topic['animalness'] +\
...                      .1*topic['cityness']
```



https://upload.wikimedia.org/wikipedia/commons/7/70/Topic_model_scheme.webm#t=00:00:01,00:00:17.600



	Doc-1	Doc-2	Doc-3	Doc-4
Topic-1	0.425	0.032	0.015	0.553
Topic-2	0.052	0.512	0.623	0.121

https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python

Let’s automate this manual procedure

You shall know a word by the company it keeps.

J. R. Firth (1957)

Topic Modeling Algorithms

- **LSA (Latent Semantic Analysis)**
- PCA (Principal Component Analysis)
- LDiA (Latent Dirichlet allocation)

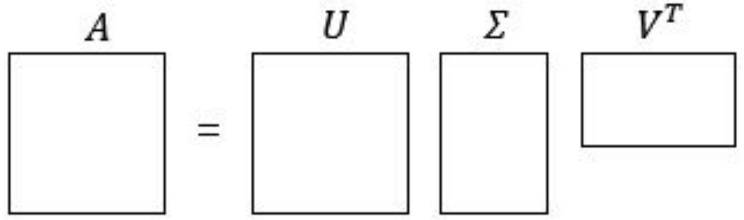
LSA (Latent Semantic Analysis)

- Uses SVD (Singular Value Decomposition) to find the combinations of words that are responsible, together, for the biggest variation in the data
- Rotate TF-IDF vectors so that the new dimensions (basis vectors, topic vectors) of the rotated vectors all align with these maximum variance directions
- Each of the dimensions (axes) becomes a combination of word frequencies rather than a single word frequency - the **weighted combinations of words** that make up “topics” used throughout the corpus
- Like the “IDF” part of TF-IDF, it tells which dimensions in the vector are important to the semantics of the documents
- SVD reorders the words and topics, to put as much of the “weight” as possible along the diagonal (https://topicmodels.west.uni-koblenz.de/ckling/tmt/svd_ap.html)
- This helps identify patterns that represent the meanings of both the topics and the words
- Algorithm should create a matrix of *n* terms X *m* topics that you can multiply by a vector of the word frequencies in a document to get your new topic vector for that document

LSA (Latent Semantic Analysis)

- We can discard those dimensions (topics) that have the least amount of variance between documents; These low-variance topics are usually noise
- If every document has roughly the same amount of some topic and that topic doesn’t help tell the documents apart, then we can get rid of it
- LSA **compresses** more meaning into fewer dimensions. We only have to retain the high-variance dimensions, the major topics that your corpus talks about in a variety of ways (with high variance). And each of these dimensions becomes your “topics,” with some weighted combination of all the words captured in each one

SVD (Singular Value Decomposition)



$$A = U\Sigma V^{\mathrm{T}}$$

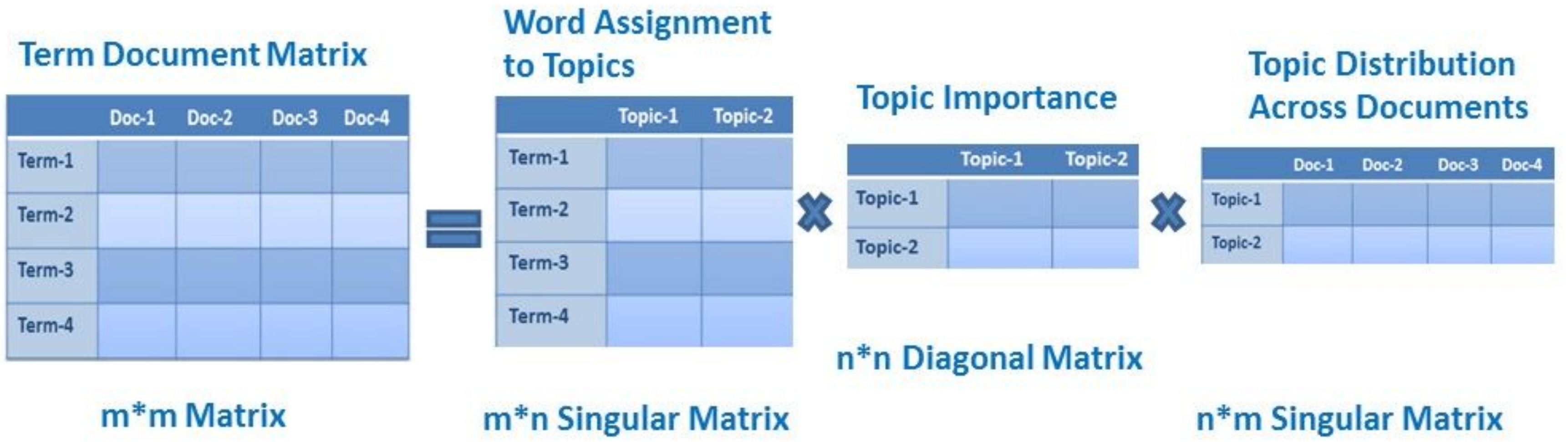
U : $m \times m$ 직교행렬 $(AA^{\mathrm{T}} = U(\Sigma\Sigma^{\mathrm{T}})U^{\mathrm{T}})$

V : $n \times n$ 직교행렬 $(A^{\mathrm{T}}A = V(\Sigma^{\mathrm{T}}\Sigma)V^{\mathrm{T}})$

Σ : $m \times n$ 직사각 대각행렬

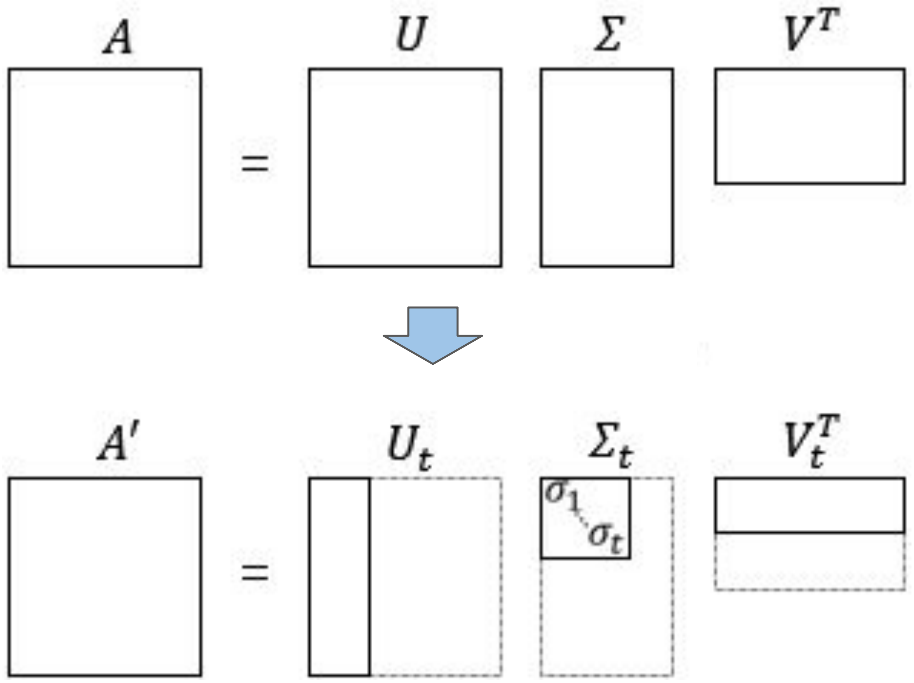
https://wikidocs.net/24949

SVD for Topic Modeling



https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python

Truncated SVD (Singular Value Decomposition)



t = # of topics

LSA Examples

- Run through a toy example at https://wikidocs.net/24949
- LSA topic modeling using sklearn.datasets.fetch_20newsgroups
- 20 topics

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
'talk.politics.misc', 'talk.religion.misc']
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med',
'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
'talk.politics.misc', 'talk.religion.misc']
```

```
Topic 1: [('like', 0.2138), ('know', 0.20031), ('people', 0.19334), ('think', 0.17802), ('good', 0.15105)]
Topic 2: [('thanks', 0.32918), ('windows', 0.29093), ('card', 0.18016), ('drive', 0.1739), ('mail', 0.15131)]
Topic 3: [('game', 0.37159), ('team', 0.32533), ('year', 0.28205), ('games', 0.25416), ('season', 0.18464)]
Topic 4: [('drive', 0.52823), ('scsi', 0.20043), ('disk', 0.15518), ('hard', 0.15511), ('card', 0.14049)]
Topic 5: [('windows', 0.40544), ('file', 0.25619), ('window', 0.1806), ('files', 0.16196), ('program', 0.14009)]
Topic 6: [('government', 0.16085), ('chip', 0.16071), ('mail', 0.15626), ('space', 0.15047), ('information', 0.13582)]
Topic 7: [('like', 0.67121), ('bike', 0.14274), ('know', 0.11189), ('chip', 0.11043), ('sounds', 0.10389)]
Topic 8: [('card', 0.44948), ('sale', 0.21639), ('video', 0.21318), ('offer', 0.14896), ('monitor', 0.1487)]
Topic 9: [('know', 0.44869), ('card', 0.35699), ('chip', 0.17169), ('video', 0.15289), ('government', 0.15069)]
Topic 10: [('good', 0.41575), ('know', 0.23137), ('time', 0.18933), ('bike', 0.11317), ('jesus', 0.09421)]
Topic 11: [('think', 0.7832), ('chip', 0.10776), ('good', 0.10613), ('thanks', 0.08985), ('clipper', 0.07882)]
Topic 12: [('thanks', 0.37279), ('right', 0.21787), ('problem', 0.2172), ('good', 0.21405), ('bike', 0.2116)]
Topic 13: [('good', 0.36691), ('people', 0.33814), ('windows', 0.28286), ('know', 0.25238), ('file', 0.18193)]
Topic 14: [('space', 0.39894), ('think', 0.23279), ('know', 0.17956), ('nasa', 0.15218), ('problem', 0.12924)]
Topic 15: [('space', 0.3092), ('good', 0.30207), ('card', 0.21615), ('people', 0.20208), ('time', 0.15716)]
Topic 16: [('people', 0.46951), ('problem', 0.20879), ('window', 0.16), ('time', 0.13873), ('game', 0.13616)]
Topic 17: [('time', 0.3419), ('bike', 0.26896), ('right', 0.26208), ('windows', 0.19632), ('file', 0.19145)]
Topic 18: [('time', 0.60079), ('problem', 0.15209), ('file', 0.13856), ('think', 0.13025), ('israel', 0.10728)]
Topic 19: [('file', 0.4489), ('need', 0.25951), ('card', 0.1876), ('files', 0.17632), ('problem', 0.1491)]
Topic 20: [('problem', 0.32797), ('file', 0.26268), ('thanks', 0.23414), ('used', 0.19339), ('space', 0.13861)]
```

Using Topic Modeling

- Understand and analyze the meaning of documents
- Reduce the representation
- Representation generalization
- Summarize the document

Applications of Topic Modeling

- **Resume Summarization**: help recruiters to evaluate resumes by a quick glance
- **Search Engine Optimization**: online articles, blogs, and documents can be tag easily by identifying the topics and associated keywords, which can improve optimize search results
- **Recommender System Optimization**: recommender systems act as an information filter and advisor according to the user profile and previous history. It can help us to discover unvisited relevant content based on past visits
- **Customer Support**: Discovering relevant topics and associated keywords in customer complaints and feedback for examples product and service specifications, department, and branch details. Such information help company to directly rotated the complaint in respective department.
- **Healthcare industry**: help extract useful and valuable information from unstructured medical reports. This information can be used for patients treatment and medical science research purpose

https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python