Theory

Wednesday, December 06, 2017 12:43 PM

What is Topic Modeling?

- **Topic Modeling** is a process to automatically identify topics (subject/theme) present in a text object (a text document such as tweet, facebook post or a collection of several documents) **and to derive** hidden patterns exhibited by a text corpus.
- Topic Modeling produces '**topics**' as outputs after processing a collection of text documents
- **Topics** = a repeating pattern of co-occurring terms in a corpus; each topic comprises a group of words.
- **Intuition**: A document could have more than 1 topic associated with it. Different topics may share some words.

What can we achieve out of Topic Modeling?

- Identify the actual topics in the collection of documents
- **Topic Distribution** for each document
- Unusual Use Case: Topic Modeling can be used as a feature selection technique for text classification
- Great Tool for Exploratory Text Analysis
- Essentially, solves Text Clustering Problem
    - We cluster words to form topics
    - We cluster documents based on the probability of occurrence of different topics

Two Approaches commonly followed for Topic Modeling:

- Probabilistic Latent Semantic Analysis (PLSA)
- Latent Dirichlet Allocation (LDA) - More recent and widely adopted!

LDA:

1. In vector space, the collection of documents can be represented as a Document - Term Matrix Document-Term or Document - Words Matrix can be divided into
   Document - Topics Matrix (dimension = N,K; N = No. of Documents; K = No. of Topics)
   Topics - Words Matrix (dimension = K,M; K= No. of Topics; M = Vocabulary Size)
   LDA uses sampling to keep improving the above 2 matrices
   For every topic among the K topics, two probabilities are created: p1 - p(topic t / document d) = the proportion of words in document d that are currently assigned to topic t = the probabily of the topic t to be present in document d
   p2 - p(word w / topic t) = the proportion of assignments to topic t over all documents that come from this word w
   = the probability of the word w to be present in topic t
   the probabilities are updated for every word w in each topic t in every document d

1. LDA is a generative model
    a. Generative vs Discriminative Model:
       A **generative algorithm** models how the data was generated in order to categorize a signal. It asks the question: based on my generation assumptions, which category is most likely to generate this signal?

       A **discriminative algorithm** does not care about how the data was generated, it simply categorizes a given signal.

       From <https://stackoverflow.com/questions/879432/what-is-the-difference-between-a-generative-and-discriminative-algorithm>

       Let's say you have input data x and you want to classify the data into labels y. A generative model learns the **joint probability distribution** `p(x,y)` and a discriminative model learns the **conditional probability distribution** `p(y|x)` - which you should read as *"the probability of y given x"*.
       Here's a really simple example. Suppose you have the following data in the form (x,y):
       `(1,0), (1,0), (2,0), (2, 1)`
       `p(x,y)` is
       ```
            y=0   y=1
           -----------
       x=1 | 1/2   0
       x=2 | 1/4   1/4
       ```
       `p(y|x)` is
       ```
            y=0   y=1
           -----------
       x=1 | 1     0
       x=2 | 1/2   1/2
       ```
       If you take a few minutes to stare at those two matrices, you will understand the difference between the two probability distributions.
       The distribution `p(y|x)` is the natural distribution for classifying a given example `x` into a class `y`, which is why algorithms that model this directly are called discriminative algorithms. Generative algorithms model `p(x,y)`, which can be tranformed into `p(y|x)` by applying Bayes rule and then used for classification. However, the distribution `p(x,y)` can also be used for other purposes. For example you could use `p(x,y)` to *generate* likely `(x,y)` pairs.
       From the description above you might be thinking that generative models are more generally useful and therefore better, but it's not as simple as that. This paper is a very popular reference on the subject of discriminative vs. generative classifiers, but it's pretty heavy going. The overall gist is that discriminative models generally outperform generative models in classification tasks.

       From <https://stackoverflow.com/questions/879432/what-is-the-difference-between-a-generative-and-discriminative-algorithm>

       Models that fall under Discriminative (logistic regression, SVMs) and Generative (Naïve Bayes)
       http://www.cedar.buffalo.edu/~srihari/CSE574/Discriminative-Generative.pdf

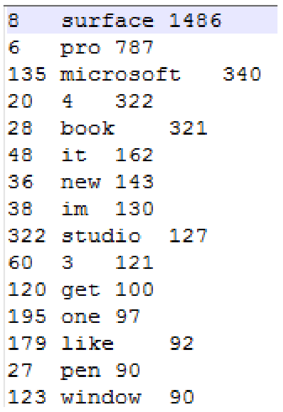
2. Limitations of Topic Modeling:
    a. Deciding on the number of topics
    b. Making sense of the topics (Topics are just word distributions, making sense of the words and generating labels is subjective)

Practical/Code oriented steps - Approach 1

Wednesday, December 06, 2017 12:44 PM

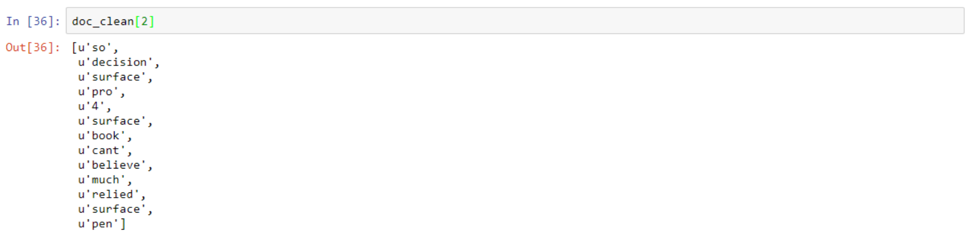
<u>Approach 1: (without using Scikit-Learn)</u>

1. Following are the steps:
    a. Pre-processing and Normalizing the document - converting into a suitable format
    b. Create a term dictionary where every unique term in the corpus is assigned an index.
        i. dictionary = corpora.Dictionary(doc_clean) #doc_clean is a list of words
        ii. dictionary.save_as_text('something.txt',sort_by_word=False)
        Gives an id or index to each unique term; also showing its frequency of its occurrence

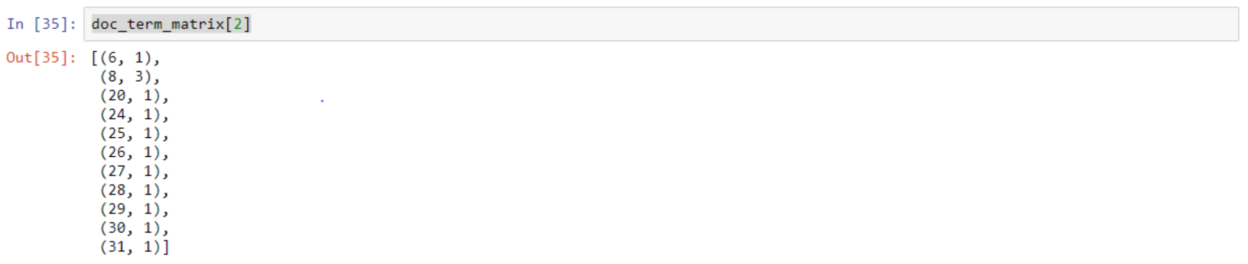


```
8   surface 1486
6   pro 787
135 microsoft   340
20  4   322
28  book    321
48  it  162
36  new 143
38  im  130
322 studio  127
60  3   121
120 get 100
195 one 97
179 like    92
27  pen 90
123 window  90
```

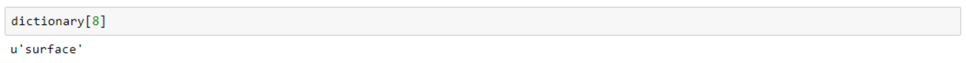
    c. Converting list of documents (doc_clean corpus) into Document Term Matrix using dictionary prepared above
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
        This creates a list of lists; each list in the overall doc_term_matrix, gives the id of the word present with its frequency



```
In [36]: doc_clean[2]

Out[36]: [u'so',
          u'decision',
          u'surface',
          u'pro',
          u'4',
          u'surface',
          u'book',
          u'cant',
          u'believe',
          u'much',
          u'relied',
          u'surface',
          u'pen']
```

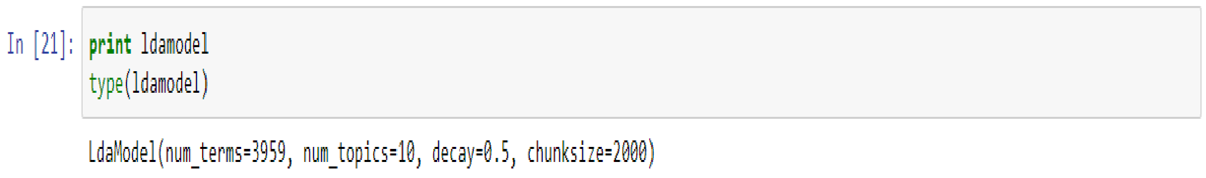


```
In [35]: doc_term_matrix[2]

Out[35]: [(6, 1),
          (8, 3),
          (20, 1),
          (24, 1),
          (25, 1),
          (26, 1),
          (27, 1),
          (28, 1),
          (29, 1),
          (30, 1),
          (31, 1)]
```

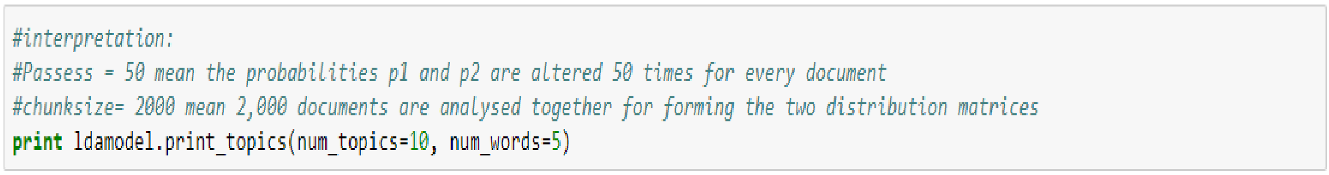


```
dictionary[8]

u'surface'
```

    d. Creating the object for LDA model
    Lda = gensim.models.ldamodel.LdaModel
    e. Running and Training LDA model on the document term matrix.
    Passes=50
    Number_of_topics=10
    ldamodel = Lda(doc_term_matrix, num_topics=Number_of_topics, id2word = dictionary, passes=Passes)
    (default chunksize = 2000)
    f.

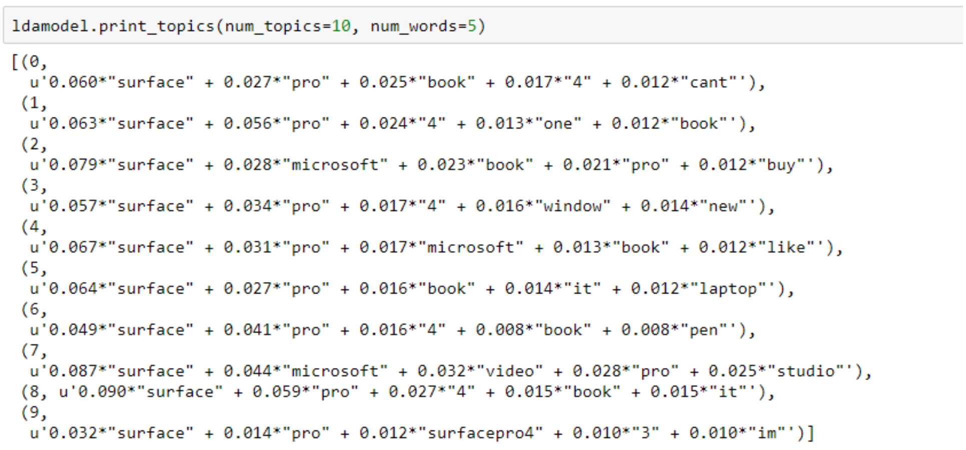


```
In [21]: print ldamodel
         type(ldamodel)

LdaModel(num_terms=3959, num_topics=10, decay=0.5, chunksize=2000)
```



```
#interpretation:
#Passess = 50 mean the probabilities p1 and p2 are altered 50 times for every document
#chunksize= 2000 mean 2,000 documents are analysed together for forming the two distribution matrices
print ldamodel.print_topics(num_topics=10, num_words=5)
```

g. While printing topics we can fix the no. of words per topic:

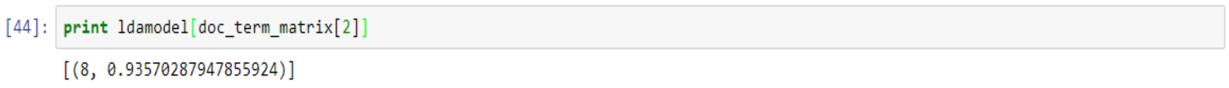


```
ldamodel.print_topics(num_topics=10, num_words=5)
```

```
[(0,
  u'0.060*"surface" + 0.027*"pro" + 0.025*"book" + 0.017*"4" + 0.012*"cant"'),
 (1,
  u'0.063*"surface" + 0.056*"pro" + 0.024*"4" + 0.013*"one" + 0.012*"book"'),
 (2,
  u'0.079*"surface" + 0.028*"microsoft" + 0.023*"book" + 0.021*"pro" + 0.012*"buy"'),
 (3,
  u'0.057*"surface" + 0.034*"pro" + 0.017*"4" + 0.016*"window" + 0.014*"new"'),
 (4,
  u'0.067*"surface" + 0.031*"pro" + 0.017*"microsoft" + 0.013*"book" + 0.012*"like"'),
 (5,
  u'0.064*"surface" + 0.027*"pro" + 0.016*"book" + 0.014*"it" + 0.012*"laptop"'),
 (6,
  u'0.049*"surface" + 0.041*"pro" + 0.016*"4" + 0.008*"book" + 0.008*"pen"'),
 (7,
  u'0.087*"surface" + 0.044*"microsoft" + 0.032*"video" + 0.028*"pro" + 0.025*"studio"'),
 (8, u'0.090*"surface" + 0.059*"pro" + 0.027*"4" + 0.015*"book" + 0.015*"it"'),
 (9,
  u'0.032*"surface" + 0.014*"pro" + 0.012*"surfacepro4" + 0.010*"3" + 0.010*"im"')]
```

h.

i. We can also determine the topic distribution of a particular document or post:
Topic 8 is the most discussed topic

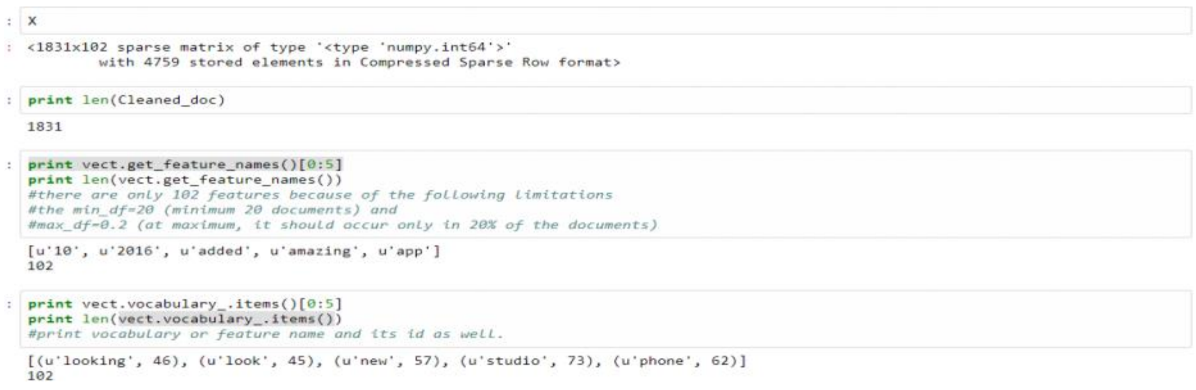


```
[44]: print ldamodel[doc_term_matrix[2]]

      [(8, 0.93570287947855924)]
```

Code Steps (Approach 2 - with Scikit-Learn)

Wednesday, December 06, 2017 3:49 PM

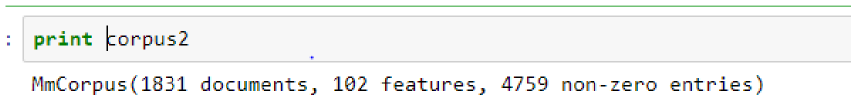
1. Pre-processing and Normalizing the document - converting into a suitable format (same as Approach 1)
2. Creating a dictionary from the vectorized form of the content
    a. Vectorize the data
    from sklearn.feature_extraction.text import CountVectorizer
    #Creating an object of the Count Vectorizer:
    vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english')
    Cleaned_doc=pd.Series(ip["Modified Text"])
    X = vect.fit_transform(Cleaned_doc) # applying the vector object on a Series of original content
    X is a sparse matrix of 1831 X 102 (no. of documents X features)



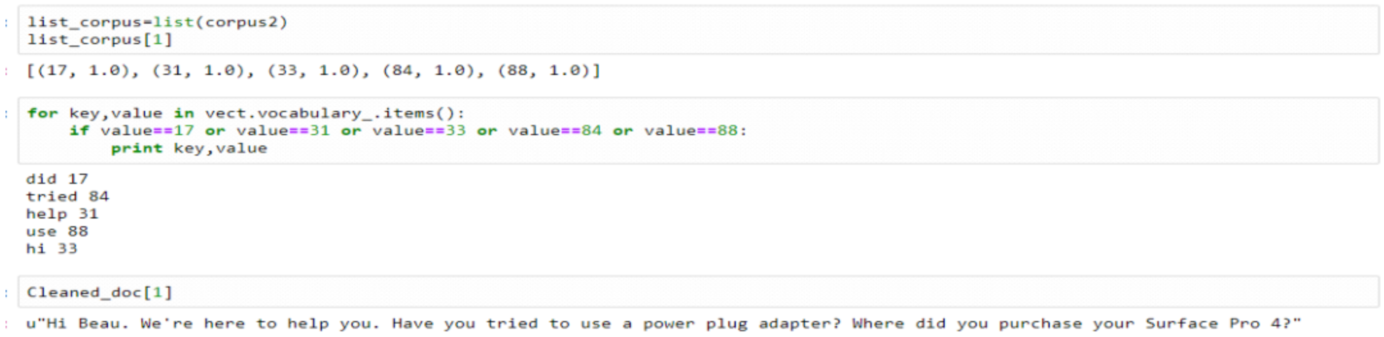
```
: X
: <1831x102 sparse matrix of type '<type 'numpy.int64'>'
          with 4759 stored elements in Compressed Sparse Row format>

: print len(Cleaned_doc)
  1831

: print vect.get_feature_names()[0:5]
  print len(vect.get_feature_names())
  #there are only 102 features because of the following limitations
  #the min_df=20 (minimum 20 documents) and
  #max_df=0.2 (at maximum, it should occur only in 20% of the documents)
  [u'10', u'2016', u'added', u'amazing', u'app']
  102

: print vect.vocabulary_.items()[0:5]
  print len(vect.vocabulary_.items())
  #print vocabulary or feature name and its id as well.
  [(u'looking', 46), (u'look', 45), (u'new', 57), (u'studio', 73), (u'phone', 62)]
  102
```

    b. # Mapping from word IDs to words (To be used in LdaModel's id2word parameter)
    id_map = dict((v, k) for k, v in vect.vocabulary_.items())
    These two steps create the 'dictionary' similar to approach 1
3. Creating a corpus similar to the document term matrix
    # Converting the sparse matrix created using scikit learn to a gensim corpus.
    corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)

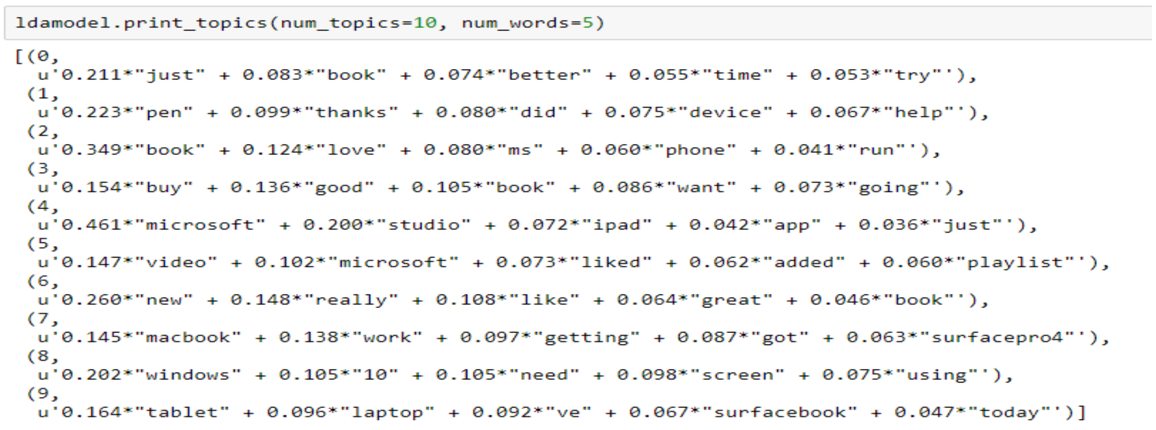
    If you have to print the gensim corpus, convert into a matrix market format corpus and see the data
    #To visibly see how the corpus is built, save the corpus in the Matrix Market format:
    corpora.MmCorpus.serialize('corpus.mm', corpus)
    #load the saved corpus
    corpus2 = corpora.MmCorpus('corpus.mm')



```
: print corpus2
  MmCorpus(1831 documents, 102 features, 4759 non-zero entries)
```



```
: list_corpus=list(corpus2)
  list_corpus[1]
: [(17, 1.0), (31, 1.0), (33, 1.0), (84, 1.0), (88, 1.0)]

: for key,value in vect.vocabulary_.items():
      if value==17 or value==31 or value==33 or value==84 or value==88:
          print key,value
  did 17
  tried 84
  help 31
  use 88
  hi 33

: Cleaned_doc[1]
: u"Hi Beau. We're here to help you. Have you tried to use a power plug adapter? Where did you purchase your Surface Pro 4?"
```



```python
ldamodel.print_topics(num_topics=10, num_words=5)
```

```
[(0,
  u'0.211*"just" + 0.083*"book" + 0.074*"better" + 0.055*"time" + 0.053*"try"'),
 (1,
  u'0.223*"pen" + 0.099*"thanks" + 0.080*"did" + 0.075*"device" + 0.067*"help"'),
 (2,
  u'0.349*"book" + 0.124*"love" + 0.080*"ms" + 0.060*"phone" + 0.041*"run"'),
 (3,
  u'0.154*"buy" + 0.136*"good" + 0.105*"book" + 0.086*"want" + 0.073*"going"'),
 (4,
  u'0.461*"microsoft" + 0.200*"studio" + 0.072*"ipad" + 0.042*"app" + 0.036*"just"'),
 (5,
  u'0.147*"video" + 0.102*"microsoft" + 0.073*"liked" + 0.062*"added" + 0.060*"playlist"'),
 (6,
  u'0.260*"new" + 0.148*"really" + 0.108*"like" + 0.064*"great" + 0.046*"book"'),
 (7,
  u'0.145*"macbook" + 0.138*"work" + 0.097*"getting" + 0.087*"got" + 0.063*"surfacepro4"'),
 (8,
  u'0.202*"windows" + 0.105*"10" + 0.105*"need" + 0.098*"screen" + 0.075*"using"'),
 (9,
  u'0.164*"tablet" + 0.096*"laptop" + 0.092*"ve" + 0.067*"surfacebook" + 0.047*"today"')]
```