# Topic Modeling Amazon Reviews

May 20, 2017

```
In [1]: from nltk.tokenize import RegexpTokenizer
        from nltk.corpus import stopwords
        from stop_words import get_stop_words
        from nltk.stem.snowball import SnowballStemmer
        from gensim import corpora, models
        import gensim
```

## 1 Loading our data

Loading the provided reviews subset CSV into a Pandas dataframe:

```
In [2]: import pandas as pd
        import numpy as np

        #df = pd.read_csv('reviews.csv')

        df = pd.read_table('reviews_2.csv',sep=',',header=None)
        df.columns = ['ID','numDate','prod','overall','helpful','votes',
        'date','asin','summary','reviewText']
```

Now that we have a nice corpus of text, let's go through some of the standard preprocessing required for almost any topic modeling or NLP problem. This approach will involve:

1. Tokenizing: converting a document to its atomic elements
2. Stopping: removing meaningless words
3. Stemming: merging words that are equivalent in meaning

## 2 Tokenization

We have many ways to segment our document into its atomic elements. To start we'll tokenize the document into words. For this instance we'll use NLTK's `tokenize.regexp` module. You can see how this works in a fun interactive way here: try 'w+' at http://regexr.com/:

```
In [3]: tokenizer = RegexpTokenizer(r'\w+')
```

Running through part of the first review to demonstrate:

```
In [4]: doc_1 = df.reviewText[0]
```

```
In [5]: # Using one of our docs as an example
        tokens = tokenizer.tokenize(doc_1.lower())

        print('{} characters in string vs {} words in a list'.format(len(doc_1),
        print(tokens[:10])
```

```
115 characters in string vs 18 words in a list
['whobbly', 'to', 'ride', 'going', 'around', 'corners', 'very',
'carefully', 'to', 'not']
```

## 3 Stop Words

Determiners like "the" and conjunctions such as "or" and "for" do not add value to the simple topic model. These types of words are known as stop words and are desired to be removed them from our list of tokens. The definition of a stop work changes depending on the context of the examined documents.

Super list of stop words from the `stop_words` and `nltk` package below.

```
merged_stopwords = [*nltk_stpwd, *stop_words_stpwd]
```

```
In [6]: nltk_stpwd = stopwords.words('english')
        stop_words_stpwd = get_stop_words('en')
        merged_stopwords = list(set(nltk_stpwd + stop_words_stpwd))

        print(len(set(merged_stopwords)))
        print(merged_stopwords[:10])
```

```
207
['i',"hadn't","you'd",'theirs','would','between','t','nor','how','more']
```

```
In [7]:
stopped_tokens=[token for token in tokens if not token in merged_stopwords]
print(stopped_tokens[:10])
```

```
['whobbly', 'ride', 'going', 'around', 'corners', 'carefully', 'tip',
'product', 'arrived', 'excellent']
```

## 4 Stemming

Stemming allows the reduction of inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance, running and runner to run. Another example:

*Amazon's catalog contains bike tires in different sizes and colors ⇒ Amazon catalog contain bike tire in differ size and color*

Stemming is a basic and crude heuristic compared to Lemmatization which understands vocabulary and morphological analysis instead of lobbing off the end of words. Essentially Lemmatization removes inflectional endings to return the word to its base or dictionary form of a word, which is defined as the lemma. Great illustrative examples from Wikipedia:

1. *The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.*
2. *The word "walk" is the base form for word "walking", and hence this is matched in both stemming and lemmatisation.*
3. *The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatisation can in principle select the appropriate lemma depending on the context.*

I start with the common Snowball stemming method, a successor of sorts of the original Porter Stemmer which is implemented in NLTK:

```
In [8]: # Instantiate a Snowball stemmer
        sb_stemmer = SnowballStemmer('english')
```

Note that p_stemmer requires all tokens to be type str. p_stemmer returns the string parameter in stemmed form, so we need to loop through our stopped_tokens:

```
In [9]: stemmed_tokens = [sb_stemmer.stem(token) for token in stopped_tokens]
        print(stemmed_tokens)
```

```
['whobbl', 'ride', 'go', 'around', 'corner', 'care', 'tip', 'product',
'arriv', 'excel', 'condit']
```

## 5 Putting together a document-term matrix

In order to create an LDA model the 3 steps from above (tokenizing, stopping, stemming) are needed together to create a list of documents (list of lists) to then generate a document-term matrix (unique terms as rows, documents or reviews as columns). This matrix will tell how frequently each term occurs with each individual document.

```
In [10]:
%%time
num_reviews = df.shape[0]
doc_set = [df.reviewText[i] for i in range(num_reviews)]
texts = []

for doc in doc_set:
        # putting our three steps together
        q_tokens = tokenizer.tokenize(word.lower())
        q_stopped_tokens = [token for token in q_tokens
                            if not token in merged_stopwords]
        q_stemmed_tokens=[sb_stemmer.stem(token) for token in q_stopped_tokens]
        # add tokens to list
        texts.append(q_stemmed_tokens[0])


Wall time: 439 ms


In [11]: print(texts[0]) # examine review 1
```

```
['whobbl', 'ride', 'go', 'around', 'corner', 'care', 'tip',
'product', 'arriv', 'excel', 'condit']
```

## 6 Transform tokenized documents into an id-term dictionary

Gensim's Dictionary method encapsulates the mapping between normalized words and their integer ids. Note a term will have an id of some number and in the subsequent bag of words step we can see that id will have a count associated with it.

```
In [12]: # Gensim's Dictionary encapsulates the mapping betweennormalized words
         texts_dict = corpora.Dictionary(texts)
         texts_dict.save('auto_review.dict') # lets save to disk for later use
         # Examine each token's unique id
         print(texts_dict)
```

```
Dictionary(2343 unique tokens: ['faster', 'thick', 'guard', 'tour', 'teas']...)
```

To see the mapping between words and their ids we can use the `token2id` method:

```
In [13]: import operator
         print("IDs 1 through 10: {}".format(sorted(texts_dict.token2id.items(),
         key=operator.itemgetter(1), reverse = False)[:10]))
```

```
IDs 1 through 10: [('tip', 0), ('corner', 1), ('product', 2),
        ('go', 3),('ride', 4), ('condit', 5), ('whobbl', 6),
        ('around', 7), ('arriv', 8), ('care', 9)]
```

```
In [14]: #Guess the original work and examine the count difference between the
        #1 most frequent term and the #10 most frequent term:

         print(df.reviewText.str.contains("balance").value_counts())
         print()
         print(df.reviewText.str.contains("lot").value_counts())
```

```
        False    634
        True      36
        Name: reviewText, dtype: int64

        False    625
        True      45
        Name: reviewText, dtype: int64
```

We have a lot of unique tokens, let's see what happens if we ignore tokens that appear in less than 30 documents or more than 15% documents. Granted this is arbitrary but a quick search shows tons of methods for reducing noise.

```
In [15]: texts_dict.filter_extremes(no_below=30, no_above=0.15) # inlace filter
         print(texts_dict)
         print("top terms:")
         print(sorted(texts_dict.token2id.items(), key=operator.itemgetter(1),
         reverse = False)[:10])
```

```
Dictionary(86 unique tokens: ['two', 'took', 'enjoy', 'wife', 'happi']...)
top terms:
[('easi', 0), ('took', 1), ('enjoy', 2), ('wife', 3), ('happi', 4),
('one', 5),('much', 6), ('nice', 7), ('time', 8), ('thank', 9)]
```

We went from **2343** unique tokens to **86** after filtering. Looking at the top 10 tokens it looks like we got more specific subjects opposed to adjectives.

## 7 Creating bag of words

Next let's turn `texts_dict` into a bag of words instead. doc2bow converts a `document` (a list of words) into the bag-of-words format (list of `(token_id, token_count)` tuples).

```
In [16]: corpus = [texts_dict.doc2bow(text) for text in texts]
         len(corpus)

Out[16]: 670
```

The corpus is 670 long, the amount of reviews in our dataset and in our dataframe. Let's dump this bag-of-words into a file to avoid parsing the entire text again:

```
In [17]: %%time
         # Matrix Market format
         https://radimrehurek.com/gensim/corpora/mmcorpus.html
         gensim.corpora.MmCorpus.serialize('amzn_auto_review.mm', corpus)

Wall time: 56.5 ms
```

## 8 Training an LDA model

Training an LDA model using our BOW corpus as training data requires a number of topics, which is set to 10. The number of passes in the training of the model is set to 100 (should be enough).

```
In [18]:
%%time
lda_model = gensim.models.LdaModel(corpus,alpha='auto',
        num_topics=10,id2word=texts_dict, passes=100)

Wall time: 1min 27s
```

## 9 Inferring Topics

Below are the top 5 words associated with 4 random topics. The float next to each word is the weight showing how much the given word influences this specific topic.

```
In [19]: # For `num_topics` number of topics,
        return `num_words` most significant words
         lda_model.show_topics(num_topics=4,num_words=5)
```

```
Out[19]: [
(0,
'0.138*"year"+0.114*"old"+0.079*"product" + 0.070*"bought" + 0.062*"perfect"'),
(1,
'0.108*"seat"+0.105*"go"+0.075*"got" + 0.062*"basket" + 0.051*"around"'),
(2,
'0.057*"trike"+0.050*"need"+0.048*"fender" + 0.040*"work" + 0.040*"tire"'),
(3,
'0.096*"took"+0.080*"time"+0.065*"new" + 0.057*"littl" + 0.052*"fun"'),
(4,
'0.127*"tricycl"+0.117*"good"+0.104*"qualiti"+0.082*"price"+0.063*"schwinn"'),
(5,
'0.091*"easi"+0.086*"use"+0.058*"realli" + 0.056*"made" + 0.052*"take"'),
(6,
'0.118*"fender"+0.107*"rear"+0.069*"box" + 0.049*"part" + 0.047*"wheel"'),
(7,
'0.100*"nice"+0.048*"want"+0.046*"happi" + 0.043*"one" + 0.043*"sturdi"'),
(8,
'0.152*"wheel"+0.057*"trike"+0.051*"3" + 0.047*"brake" + 0.042*"difficult"'),
(9,
'0.142*"make"+0.128*"thank"+0.080*"easi" + 0.077*"shop" + 0.059*"got"')]
```

LDA is a probabilistic mixture of mixtures (or admixture) model for grouped data. The observed data (words) within the groups (documents) are the result of probabilistically choosing words from a specific topic (multinomial over the vocabulary), where the topic is itself drawn from a document-specific multinomial that has a global Dirichlet prior. This means that words can belong to various topics in various degrees.

## 10 Querying the LDA Model

Pass an arbitrary string to our model and evaluate what topics are most associated with it.

```
In [20]:
raw_query = 'issue'

query_words = raw_query.split()
query = []
for word in query_words:
        # ad-hoc reuse steps from above
        q_tokens = tokenizer.tokenize(word.lower())
        q_stopped_tokens = [word for word in q_tokens if not word in merged_stop
        q_stemmed_tokens = [sb_stemmer.stem(word) for word in q_stopped_tokens]
        query.append(q_stemmed_tokens[0])
        #different from above, this is not a lists of lists!

print(query)
```

```
['issu']
```

```
In [21]: # translate words in query to ids and frequencies.
         id2word = gensim.corpora.Dictionary()
         _ = id2word.merge_with(texts_dict) # garbage
```

```
In [22]: # translate this document into (word, frequency) pairs
         query = id2word.doc2bow(query)
         print(query)
```

```
[(18, 1)]
```

If we run this constructed query against our trained mode we will get each topic and the likelihood that the `query` relates to that topic. Remember we arbitrarily specified 4 topics when we made the model. When we organize this list to find the most relative topics, we see some intuitive results.

```
In [23]: a = list(sorted(lda_model[query], key=lambda x: x[1],reverse=True))
# sort by the second entry in the tuple
         a
```

```
Out[23]: [(3, 0.66089604585589257),
 (0, 0.042840632894293761),
 (7, 0.041895896144469633),
 (2, 0.039156986576921418),
 (4, 0.038719777833920874),
 (5, 0.038465777946288152),
 (1, 0.038185859271443023),
 (8, 0.035220474260673415),
 (6, 0.03455538873028282),
 (9, 0.030063160485814349)]
```

```
In [24]: lda_model.print_topic(a[0][0]) #most related
```

```
Out[24]: '0.096*"took" + 0.080*"time" + 0.065*"new" + 0.057*"littl" +
         0.052*"fun" + 0.048*"hour" + 0.048*"husband" + 0.042*"two" +
         0.042*"like" + 0.042*"wife"'
```

```
In [25]: lda_model.print_topic(a[-1][0]) #least related
```

```
Out[25]: '0.142*"make" + 0.128*"thank" + 0.080*"easi" + 0.077*"shop" +
         0.059*"got" + 0.045*"take" + 0.043*"back" + 0.040*"pedal" +
         0.033*"handl" + 0.029*"littl"'
```

```
In [26]: import pandas as pd
         word_least = pd.DataFrame(lda_model.show_topic(a[-1][0]),
         columns=['words','count'])
         word_most = pd.DataFrame(lda_model.show_topic(a[0][0]),
         columns=['words','count'])
```

```
In [27]: %matplotlib inline
         import matplotlib.pyplot as plt
         word_least.plot(kind='bar',color='#EE4266',
                 x=word_least['words'], legend=False,
                 title='Words of the Less Related Topic about "Issues"',
                figsize=(10,6))
         word_most.plot(kind='bar',color='#5DFDCB',
                 x=word_most['words'],legend=False,
                 title='Words of the Most Related Topic about "Issues"',
                 figsize=(10,6))
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2c6c39e7978>
```

## 11 Conclusion

Our query "issues" seems to be highly related to the topic number 3, with a correlation of 73%. The rest of topic are mostly unrelated, all below 10% correlation.

By taking these inferred topics and analyzing the sentiment of their corresponding documents (reviews) we can find out what customers are saying (or feeling) about specific products. The LDA model can extract representative statements or quotes, enabling us to summarize customers' opinions about products, perhaps even displaying them on the site. LDA models groups of customers to topics which are groups of products that frequently occur within some customer's orders over time.