topic_methods

July 2, 2018

1 New Term Topics Methods and Document Coloring

```
In [1]: from gensim.corpora import Dictionary
        from gensim.models import ldamodel
        import numpy
        %matplotlib inline
```

```
/usr/local/lib/python2.7/site-packages/matplotlib/font_manager.py:273: UserWarning:
Matplotlib is building the font cache using fc-list. This may take a moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may take a
moment.')
```

We're setting up our corpus now. We want to show off the new `get_term_topics` and `get_document_topics` functionalities, and a good way to do so is to play around with words which might have different meanings in different context.

The word `bank` is a good candidate here, where it can mean either the financial institution or a river bank. In the toy corpus presented, there are 11 documents, 5 `river` related and 6 `finance` related.

```
In [2]: texts = [['bank','river','shore','water'],
                 ['river','water','flow','fast','tree'],
                 ['bank','water','fall','flow'],
                 ['bank','bank','water','rain','river'],
                 ['river','water','mud','tree'],
                 ['money','transaction','bank','finance'],
                 ['bank','borrow','money'],
                 ['bank','finance'],
                 ['finance','money','sell','bank'],
                 ['borrow','sell'],
                 ['bank','loan','sell']]

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

We set up the LDA model in the corpus. We set the number of topics to be 2, and expect to see one which is to do with river banks, and one to do with financial banks.

```
In [3]: numpy.random.seed(1) # setting random seed to get the same results each time.
        model = ldamodel.LdaModel(corpus, id2word=dictionary, num_topics=2)
```

```
In [4]: model.show_topics()
```

```
Out[4]: [(0,
          u'0.164*"bank" + 0.142*"water" + 0.108*"river" + 0.076*"flow" + 0.067*"borrow" + 0.06
         (1,
          u'0.196*"bank" + 0.120*"finance" + 0.100*"money" + 0.082*"sell" + 0.067*"river" + 0.0
```

And like we expected, the LDA model has given us near perfect results. Bank is the most influential word in both the topics, as we can see. The other words help define what kind of bank we are talking about. Let's now see where our new methods fit in.

1.0.1 get_term_topics

The function `get_term_topics` returns the odds of that particular word belonging to a particular topic. A few examples:

```
In [5]: model.get_term_topics('water')
```

```
Out[5]: [(0, 0.12821234071249418), (1, 0.047247458568794511)]
```

Makes sense, the value for it belonging to `topic_0` is a lot more.

```
In [6]: model.get_term_topics('finance')
```

```
Out[6]: [(0, 0.017179349495865623), (1, 0.10331511184214655)]
```

This also works out well, the word finance is more likely to be in topic_1 to do with financial banks.

```
In [7]: model.get_term_topics('bank')
```

```
Out[7]: [(0, 0.15042435080542094), (1, 0.18044627232201182)]
```

And this is particularly interesting. Since the word bank is likely to be in both the topics, the values returned are also very similar.

1.0.2 get_document_topics and Document Word-Topic Coloring

`get_document_topics` is an already existing gensim functionality which uses the `inference` function to get the sufficient statistics and figure out the topic distribution of the document.

The addition to this is the ability for us to now know the topic distribution for each word in the document. Let us test this with two different documents which have the word bank in it, one in the finance context and one in the river context.

The `get_document_topics` method returns (along with the standard document topic proprtion) the word_type followed by a list sorted with the most likely topic ids, when `per_word_topics` is set as true.

```
In [8]: bow_water = ['bank','water','bank']
        bow_finance = ['bank','finance','bank']
```

```
In [9]: bow = model.id2word.doc2bow(bow_water) # convert to bag of words format first
        doc_topics, word_topics, phi_values = model.get_document_topics(bow,
        per_word_topics=True)

        word_topics
```

```
Out[9]: [(0, [0, 1]), (3, [0, 1])]
```

Now what does that output mean? It means that like `word_type 1`, our `word_type 3`, which is the word `bank`, is more likely to be in `topic_0` than `topic_1`.

You must have noticed that while we unpacked into `doc_topics` and `word_topics`, there is another variable - `phi_values`. Like the name suggests, phi_values contains the phi values for each topic for that particular word, scaled by feature length. Phi is essentially the probability of that word in that document belonging to a particular topic. The next few lines should illustrate this.

```
In [10]: phi_values
```

```
Out[10]: [(0, [(0, 0.92486455564294345), (1, 0.075135444357056574)]),
          (3, [(0, 1.5817120973072454), (1, 0.41828790269275457)])]
```

This means that `word_type 0` has the following phi_values for each of the topics. What is intresting to note is `word_type 3` - because it has 2 occurences (i.e, the word `bank` appears twice in the bow), we can see that the scaling by feature length is very evident. The sum of the phi_values is 2, and not 1.

Now that we know exactly what `get_document_topics` does, let us now do the same with our second document, `bow_finance`.

```
In [11]: bow = model.id2word.doc2bow(bow_finance) # convert to bag of words format first
         doc_topics, word_topics, phi_values = model.get_document_topics(bow,
         per_word_topics=True)

         word_topics
```

```
Out[11]: [(3, [1, 0]), (12, [1, 0])]
```

And lo and behold, because the word bank is now used in the financial context, it immedietly swaps to being more likely associated with `topic_1`.

We've seen quite clearly that based on the context, the most likely topic associated with a word can change. This differs from our previous method, `get_term_topics`, where it is a 'static' topic distribution.

It must also be noted that because the gensim implementation of LDA uses Variational Bayes sampling, a `word_type` in a document is only given one topic distribution. For example, the sentence 'the bank by the river bank' is likely to be assigned to `topic_0`, and each of the bank word instances have the same distribution.

get_document_topics for entire corpus You can get `doc_topics`, `word_topics` and `phi_values` for all the documents in the corpus in the following manner :

```
In [12]: all_topics = model.get_document_topics(corpus, per_word_topics=True)

         for doc_topics, word_topics, phi_values in all_topics:
             print('New Document \n')
             print 'Document topics:', doc_topics
             print 'Word topics:', word_topics
             print 'Phi values:', phi_values
             print(" ")
             print('-------------- \n')
```

```
New Document

Document topics: [(0, 0.83270647275828524), (1, 0.16729352724171473)]
```

```
Word topics: [(0, [0, 1]), (1, [0, 1]), (2, [0, 1]), (3, [0, 1])]
Phi values: [(0, [(0, 0.96021858877561717), (1, 0.039781411224382883)]), (1, [(0,
0.87921979686273788), (1, 0.12078020313726225)]), (2, [(0, 0.94364164103826909), (1,
0.056358358961730845)]), (3, [(0, 0.88116401400740607), (1, 0.11883598599259393)])]

--------------

New Document

Document topics: [(0, 0.90379559943992582), (1, 0.096204400560074191)]
Word topics: [(0, [0, 1]), (2, [0, 1]), (4, [0]), (5, [0, 1]), (6, [0])]
Phi values: [(0, [(0, 0.98551395531215857), (1, 0.014486044687841437)]), (2, [(0,
0.97924982750620249), (1, 0.020750172493797691)]), (4, [(0, 0.99280849901823975)]),
(5, [(0, 0.97529774122781787), (1, 0.024702258772182122)]), (6, [(0,
0.99004205057244832)])]

--------------

New Document

Document topics: [(0, 0.87507219282484316), (1, 0.12492780717515681)]
Word topics: [(0, [0, 1]), (3, [0, 1]), (4, [0, 1]), (7, [0, 1])]
Phi values: [(0, [(0, 0.9783234200583657), (1, 0.021676579941634355)]), (3, [(0,
0.93272653621872503), (1, 0.067273463781275009)]), (4, [(0, 0.98919912227661466), (1,
0.010800877723385368)]), (7, [(0, 0.97541896333079636), (1, 0.024581036669203641)])]

--------------

New Document

Document topics: [(0, 0.87853819958920043), (1, 0.12146180041079958)]
Word topics: [(0, [0, 1]), (2, [0, 1]), (3, [0, 1]), (8, [0, 1])]
Phi values: [(0, [(0, 0.97596134249481492), (1, 0.024038657505185138)]), (2, [(0,
0.96571015226994938), (1, 0.034289847730050525)]), (3, [(0, 1.851545575505376), (1,
0.14845442449462365)]), (8, [(0, 0.97848202469975276), (1, 0.021517975300247363)])]

--------------

New Document

Document topics: [(0, 0.85644637406235502), (1, 0.14355362593764506)]
Word topics: [(0, [0, 1]), (2, [0, 1]), (5, [0, 1]), (9, [0, 1])]
Phi values: [(0, [(0, 0.97074863890671426), (1, 0.029251361093285893)]), (2, [(0,
0.95836933362205601), (1, 0.041630666377943965)]), (5, [(0, 0.95064079648593469), (1,
0.049359203514065378)]), (9, [(0, 0.90303582762229051), (1, 0.096964172377709504)])]

--------------

New Document

Document topics: [(0, 0.11549963646117178), (1, 0.88450036353882822)]
Word topics: [(3, [1, 0]), (10, [1, 0]), (11, [1]), (12, [1])]
Phi values: [(3, [(0, 0.040062133454181546), (1, 0.95993786654581814)]), (10, [(0,
0.020103806467996775), (1, 0.97989619353200308)]), (11, [(1, 0.9910494032913304)]),
(12, [(1, 0.99174412290358549)])]

--------------

New Document
```

```
Document topics: [(0, 0.44388593146078198), (1, 0.55611406853921797)]
Word topics: [(3, [1, 0]), (10, [1, 0]), (13, [0, 1])]
Phi values: [(3, [(0, 0.38381806344612579), (1, 0.6161819365538742 1)]), (10, [(0,
0.23442811582700812), (1, 0.76557188417299193)]), (13, [(0, 0.65651736899869417), (1,
0.34348263100130588)])]

--------------

New Document

Document topics: [(0, 0.20199255912939526), (1, 0.7980074408706047 1)]
Word topics: [(3, [1, 0]), (12, [1, 0])]
Phi values: [(3, [(0, 0.086998287940481228), (1, 0.9130017120595186 3)]), (12, [(0,
0.018652395463982233), (1, 0.98134760453601788)])]

--------------

New Document

Document topics: [(0, 0.12505726157782684), (1, 0.8749427384221732 9)]
Word topics: [(3, [1, 0]), (10, [1, 0]), (12, [1]), (14, [1, 0])]
Phi values: [(3, [(0, 0.047837589620218293), (1, 0.9521624103797816 7)]), (10, [(0,
0.024102914052397447), (1, 0.9758970859476026)]), (12, [(1, 0.99007797561579536)]),
(14, [(0, 0.04309284551399737), (1, 0.9569071544860027 2)])]

--------------

New Document

Document topics: [(0, 0.72319610071601925), (1, 0.2768038992839806 9)]
Word topics: [(13, [0, 1]), (14, [0, 1])]
Phi values: [(13, [(0, 0.91396121153662691), (1, 0.086038788463373025)]), (14, [(0,
0.75627751890079997), (1, 0.24372248109919997)])]

--------------

New Document

Document topics: [(0, 0.16978578818257647), (1, 0.8302142118174235)]
Word topics: [(3, [1, 0]), (14, [1, 0]), (15, [1, 0])]
Phi values: [(3, [(0, 0.075528355267193981), (1, 0.9244716447328059 6)]), (14, [(0,
0.068233937712710399), (1, 0.93176606228728964)]), (15, [(0, 0.035035615878295796),
(1, 0.96496438412170416)])]

--------------
```

In case you want to store `doc_topics`, `word_topics` and `phi_values` for all the documents in the corpus in a variable and later access details of a particular document using its index, it can be done in the following manner:

```
In [13]: topics = model.get_document_topics(corpus, per_word_topics=True)
         all_topics = [(doc_topics, word_topics, word_phis) for doc_topics, word_topics,
         word_phis in topics]
```

Now, I can access details of a particular document, say Document #3, as follows:

```
In [14]: doc_topic, word_topics, phi_values = all_topics[2]
         print 'Document topic:', doc_topics, "\n"
         print 'Word topic:', word_topics, "\n"
         print 'Phi value:', phi_values
```

```
Document topic: [(0, 0.1697726556081923), (1, 0.83022734439180768)]

Word topic: [(0, [0, 1]), (3, [0, 1]), (4, [0, 1]), (7, [0, 1])]

Phi value: [(0, [(0, 0.978328710597138), (1, 0.021671289402862035)]), (3, [(0,
0.93274219037812456), (1, 0.067257809621875539)]), (4, [(0, 0.98920178771276146), (1,
0.010798212287238563)]), (7, [(0, 0.97542494494492515), (1, 0.02457505505507478)])]
```

We can print details for all the documents (as shown above), in the following manner:

```
In [15]: for doc in all_topics:
             print('New Document \n')
             print 'Document topic:', doc[0]
             print 'Word topic:', doc[1]
             print 'Phi value:', doc[2]
             print(" ")
             print('-------------- \n')
```

```
New Document

Document topic: [(0, 0.83271544885346738), (1, 0.16728455114653268)]
Word topic: [(0, [0, 1]), (1, [0, 1]), (2, [0, 1]), (3, [0, 1])]
Phi value: [(0, [(0, 0.96022273559375526), (1, 0.039777264406244746)]), (1, [(0,
0.87923132506871871), (1, 0.12076867493128131)]), (2, [(0, 0.94364741442849287), (1,
0.056352585571507234)]), (3, [(0, 0.8811753817216651), (1, 0.11882461827833496)])]

--------------

New Document

Document topic: [(0, 0.90379650157173907), (1, 0.096203498428260883)]
Word topic: [(0, [0, 1]), (2, [0, 1]), (4, [0]), (5, [0, 1]), (6, [0])]
Phi value: [(0, [(0, 0.98551427047222728), (1, 0.014485729527772766)]), (2, [(0,
0.9792502760799594), (1, 0.020749723920040618)]), (4, [(0, 0.99280865663541784)]), (5,
[(0, 0.97529827308199035), (1, 0.024701726918009728)]), (6, [(0,
0.99004226821414432)])]

--------------

New Document

Document topic: [(0, 0.87508582200973173), (1, 0.12491417799026834)]
Word topic: [(0, [0, 1]), (3, [0, 1]), (4, [0, 1]), (7, [0, 1])]
Phi value: [(0, [(0, 0.978328710597138), (1, 0.021671289402862035)]), (3, [(0,
0.93274219037812456), (1, 0.067257809621875539)]), (4, [(0, 0.98920178771276146), (1,
0.010798212287238563)]), (7, [(0, 0.97542494494492515), (1, 0.02457505505507478)])]

--------------

New Document

Document topic: [(0, 0.87849699952437466), (1, 0.12150300047562536)]
Word topic: [(0, [0, 1]), (2, [0, 1]), (3, [0, 1]), (8, [0, 1])]
Phi value: [(0, [(0, 0.97594470848740367), (1, 0.024055291512596246)]), (2, [(0,
```

0.96568667415296994), (1, 0.034313325847029875)]), (3, [(0, 1.8514481357538264), (1,
0.14855186424617364)]), (8, [(0, 0.97846709644293861), (1, 0.021532903557061403)])]

New Document

Document topic: [(0, 0.85642628246505548), (1, 0.14357371753494441)]
Word topic: [(0, [0, 1]), (2, [0, 1]), (5, [0, 1]), (9, [0, 1])]
Phi value: [(0, [(0, 0.97074011917537684), (1, 0.02925988082462316)]), (2, [(0,
0.95835736297327923), (1, 0.041642637026720823)]), (5, [(0, 0.95062671803078924), (1,
0.049373281969210848)]), (9, [(0, 0.90300955638764013), (1, 0.096990443612359839)])]

New Document

Document topic: [(0, 0.11553980471363219), (1, 0.88446019528636788)]
Word topic: [(3, [1, 0]), (10, [1, 0]), (11, [1]), (12, [1])]
Phi value: [(3, [(0, 0.040094010013352048), (1, 0.95990598998664811)]), (10, [(0,
0.020120135475541409), (1, 0.97987986452445841)]), (11, [(1, 0.9910420504916706)]),
(12, [(1, 0.99173733604900283)])]

New Document

Document topic: [(0, 0.44387392752172899), (1, 0.55612607247827095)]
Word topic: [(3, [1, 0]), (10, [1, 0]), (13, [0, 1])]
Phi value: [(3, [(0, 0.38380312832253366), (1, 0.61619687167746628)]), (10, [(0,
0.23441678227547744), (1, 0.76558321772452254)]), (13, [(0, 0.65650312824652535), (1,
0.34349687175347449)])]

New Document

Document topic: [(0, 0.20190467603529849), (1, 0.79809532396470151)]
Word topic: [(3, [1, 0]), (12, [1, 0])]
Phi value: [(3, [(0, 0.086912561161162485), (1, 0.91308743883883758)]), (12, [(0,
0.018632641254402959), (1, 0.98136735874559722)])]

New Document

Document topic: [(0, 0.12500947583350866), (1, 0.87499052416649137)]
Word topic: [(3, [1, 0]), (10, [1, 0]), (12, [1]), (14, [1, 0])]
Phi value: [(3, [(0, 0.04779781295574477), (1, 0.95220218704425519)]), (10, [(0,
0.024082373476646459), (1, 0.97591762652335345)]), (12, [(1, 0.99008655395768441)]),
(14, [(0, 0.043056835672030759), (1, 0.9569431643279692 2)])]

New Document

Document topic: [(0, 0.72327037334816691), (1, 0.27672962665183315)]
Word topic: [(13, [0, 1]), (14, [0, 1])]
Phi value: [(13, [(0, 0.91400946720135656), (1, 0.085990532798643632)]), (14, [(0,

```
0.7563906403756806), (1, 0.2436093596243194)])]

--------------

New Document

Document topic: [(0, 0.1697726556081923), (1, 0.83022734439180768)]
Word topic: [(3, [1, 0]), (14, [1, 0]), (15, [1, 0])]
Phi value: [(3, [(0, 0.075516159640739211), (1, 0.9244838403592609)]), (14, [(0,
0.068222833001706978), (1, 0.93177716699829294)]), (15, [(0, 0.035029710897900725),
(1, 0.96497028910209925)])]

--------------
```

1.1 Coloring topic-terms

These methods can come in handy when we want to color the words in a corpus or a document. If we wish to color the words in a corpus (i.e, color all the words in the dictionary of the corpus), then `get_term_topics` would be a better choice. If not, `get_document_topics` would do the trick.

We'll now attempt to color these words and plot it using `matplotlib`. This is just one way to go about plotting words - there are more and better ways.

WordCloud is such a python package which also does this.

For our simple illustration, let's keep `topic_1` as red, and `topic_0` as blue.

```
In [16]: # this is a sample method to color words. Like mentioned before, there are many ways to
         do this.

         def color_words(model, doc):
             import matplotlib.pyplot as plt
             import matplotlib.patches as patches

             # make into bag of words
             doc = model.id2word.doc2bow(doc)
             # get word_topics
             doc_topics, word_topics, phi_values = model.get_document_topics(doc,
         per_word_topics=True)

             # color-topic matching
             topic_colors = { 1:'red', 0:'blue'}

             # set up fig to plot
             fig = plt.figure()
             ax = fig.add_axes([0,0,1,1])

             # a sort of hack to make sure the words are well spaced out.
             word_pos = 1/len(doc)

             # use matplotlib to plot words
             for word, topics in word_topics:
                 ax.text(word_pos, 0.8, model.id2word[word],
                         horizontalalignment='center',
                         verticalalignment='center',
                         fontsize=20, color=topic_colors[topics[0]],  # choose just the most
         likely topic
                         transform=ax.transAxes)
                 word_pos += 0.2 # to move the word for the next iter

             ax.set_axis_off()
             plt.show()
```

Let us revisit our old examples to show some examples of document coloring

```
In [17]: # our river bank document

         bow_water = ['bank','water','bank']
         color_words(model, bow_water)
```


```
In [18]: bow_finance = ['bank','finance','bank']
         color_words(model, bow_finance)
```

What is fun to note here is that while bank was colored blue in our first example, it is now red because of the financial context - something which the numbers proved to us before.

```
In [19]: # sample doc with a somewhat even distribution of words among the likely topics

         doc = ['bank', 'water', 'bank', 'finance', 'money','sell','river','fast','tree']
         color_words(model, doc)
```

We see that the document word coloring is done just the way we expected. :)

1.2 Word-coloring a dictionary

We can do the same for the entire vocabulary, statically. The only difference would be in using `get_term_topics`, and iterating over the dictionary.

We will use a modified version of the coloring code when passing an entire dictionary.

```
In [20]: def color_words_dict(model, dictionary):
             import matplotlib.pyplot as plt
             import matplotlib.patches as patches

             word_topics = []
             for word_id in dictionary:
                 word = str(dictionary[word_id])
                 # get_term_topics returns static topics, as mentioned before
                 probs = model.get_term_topics(word)
                 # we are creating word_topics which is similar to the one created by
         get_document_topics
                 try:
                     if probs[0][1] >= probs[1][1]:
                         word_topics.append((word_id, [0, 1]))
                     else:
                         word_topics.append((word_id, [1, 0]))
                 # this in the case only one topic is returned
                 except IndexError:
                     word_topics.append((word_id, [probs[0][0]]))

             # color-topic matching
             topic_colors = { 1:'red', 0:'blue'}

             # set up fig to plot
             fig = plt.figure()
             ax = fig.add_axes([0,0,1,1])

             # a sort of hack to make sure the words are well spaced out.
             word_pos = 1/len(doc)

             # use matplotlib to plot words
             for word, topics in word_topics:
                 ax.text(word_pos, 0.8, model.id2word[word],
                         horizontalalignment='center',
                         verticalalignment='center',
                         fontsize=20, color=topic_colors[topics[0]],  # choose just the most
         likely topic
                         transform=ax.transAxes)
                 word_pos += 0.2 # to move the word for the next iter

             ax.set_axis_off()
             plt.show()

In [21]: color_words_dict(model, dictionary)
```

As we can see, the red words are to do with finance, and the blue ones are to do with water.

You can also notice that some words, like mud, shore and borrow seem to be incorrectly colored - however, they are correctly colored according to the LDA model used for coloring. A small corpus means that the LDA algorithm might not assign 'ideal' topic proportions to each word. Fine tuning the model and having a larger corpus would improve the model, and improve the results of the word coloring.