**Getting started with Latent Dirichlet Allocation in Python**

<http://chrisstrelioff.ws/sandbox/2014/11/13/getting_started_with_latent_dirichlet_allocation_in_python.html>

**Installing lda**

Briefly, there are two approaches I will mention:

* **Method 1**:

I will install [**lda**](https://github.com/ariddell/lda) as a *user*

$ pip install --user lda

This will also install the required **pbr** package. Now I will have [**lda**](https://github.com/ariddell/lda) available in a setup with all the other packages I have previously installed (again, see above). With this method, you should get something like this after install:

$ pip show lda

---

Name: lda

Version: 0.3.2

Location: /home/cstrelioff/.local/lib/python2.7/site-packages

Requires: pbr, numpy

I already had **numpy** installed, so it was not modified.

..lots of numpy compilation...

In this case, **pip** will show the installation in the location specified for your virtualenvs. For me, this looks like:

(lda\_env)$ pip show lda

---

Name: lda

Version: 0.3.2

Location: /home/cstrelioff/virtenvs/lda\_env/lib/python2.7/site-packages

Requires: pbr, numpy

**An Example**

The example at the [**lda**](https://github.com/ariddell/lda) github repository looks at corpus of Reuters news releases– let’s replicate this and add some details to better understand what is going on. A script containing all of the code to follow, called ex002\_lda.py, is available at [**this gist**](https://gist.github.com/cstrelioff/38b1d16a1253c962b7d7). To get started, we do some imports:

**from** **\_\_future\_\_** **import** division, print\_function

**import** **numpy** **as** **np**

**import** **lda**

**import** **lda.datasets**

Next, we import the data used for the example. This is included with the [**lda**](https://github.com/ariddell/lda) package, so this step is simple (I also print out the data type and size for each item):

*# document-term matrix*

X = lda.datasets.load\_reuters()

**print**("type(X): {}".format(type(X)))

**print**("shape: {}**\n**".format(X.shape))

*# the vocab*

vocab = lda.datasets.load\_reuters\_vocab()

**print**("type(vocab): {}".format(type(vocab)))

**print**("len(vocab): {}**\n**".format(len(vocab)))

*# titles for each story*

titles = lda.datasets.load\_reuters\_titles()

**print**("type(titles): {}".format(type(titles)))

**print**("len(titles): {}**\n**".format(len(titles)))

type(X): <type 'numpy.ndarray'>

shape: (395, 4258)

type(vocab): <type 'tuple'>

len(vocab): 4258

type(titles): <type 'tuple'>

len(titles): 395

From the above we can see that there are 395 news items (documents) and a vocabulary of size 4258. The document-term matrix, X, has a count of the number of occurences of each of the 4258 vocabulary words for each of the 395 documents. For example, X[0,3117] is the number of times that word 3117 occurs in document 0. We can find out the count and the word that this corresponds to using (let’s also get the document title):

doc\_id = 0

word\_id = 3117

**print**("doc id: {} word id: {}".format(doc\_id, word\_id))

**print**("-- count: {}".format(X[doc\_id, word\_id]))

**print**("-- word : {}".format(vocab[word\_id]))

**print**("-- doc : {}".format(titles[doc\_id]))

doc id: 0 word id: 3117

-- count: 2

-- word : heir-to-the-throne

-- doc : 0 UK: Prince Charles spearheads British royal revolution. LONDON

1996-08-20

Of course we should expect that there are lots of zeros in the X matrix– I chose this example to get a non-zero result.

**Fiting the model**

Next we initialize and fit the LDA model. To do this we have to choose the number of topics (other methods can attempt to find the number of topics as well, but for LDA we have to assume a number). Continuing with the example we choose:

model = lda.LDA(n\_topics=20, n\_iter=500, random\_state=1)

model.fit(X)

There are a couple of parameters for the prior that we leave at the default values. As far as I can tell, this only uses symmetric priors – I’ll have to look into this more (see [**Wallach etal 2009**](http://papers.nips.cc/paper/3854-rethinking-lda-why-priors-matter) for a discussion of this issue).

**Topic-Word**

From the fit model we can look at the topic-word probabilities:

topic\_word = model.topic\_word\_

**print**("type(topic\_word): {}".format(type(topic\_word)))

**print**("shape: {}".format(topic\_word.shape))

type(topic\_word): <type 'numpy.ndarray'>

shape: (20, 4258)

From the size of the output we can see that we have a distribution over the 4258 words in the vocabulary for each of the 20 topics. For each topic, the probabilities of the words should be normalized. Let’s check the first 5:

**for** n **in** range(5):

sum\_pr = sum(topic\_word[n,:])

**print**("topic: {} sum: {}".format(n, sum\_pr))

topic: 0 sum: 1.0

topic: 1 sum: 1.0

topic: 2 sum: 1.0

topic: 3 sum: 1.0

topic: 4 sum: 1.0

We can also get the top 5 words for each topic (by probability):

n = 5

**for** i, topic\_dist **in** enumerate(topic\_word):

topic\_words = np.array(vocab)[np.argsort(topic\_dist)][:-(n+1):-1]

**print**('\*Topic {}**\n**- {}'.format(i, ' '.join(topic\_words)))

\*Topic 0

- church people told years last

\*Topic 1

- elvis music fans york show

\*Topic 2

- pope trip mass vatican poland

\*Topic 3

- film french against france festival

\*Topic 4

- king michael romania president first

\*Topic 5

- police family versace miami cunanan

\*Topic 6

- germany german war political government

\*Topic 7

- harriman u.s clinton churchill ambassador

\*Topic 8

- yeltsin russian russia president kremlin

\*Topic 9

- prince queen bowles church king

\*Topic 10

- simpson million years south irish

\*Topic 11

- charles diana parker camilla marriage

\*Topic 12

- east peace prize president award

\*Topic 13

- order nuns india successor election

\*Topic 14

- pope vatican hospital surgery rome

\*Topic 15

- mother teresa heart calcutta missionaries

\*Topic 16

- bernardin cardinal cancer church life

\*Topic 17

- died funeral church city death

\*Topic 18

- museum kennedy cultural city culture

\*Topic 19

- art exhibition century city tour

This gives us some sense of what the 20 topics might actually mean – can you see the patterns?

**Document-Topic**

The other information we get from the model is document-topic probabilities:

doc\_topic = model.doc\_topic\_

**print**("type(doc\_topic): {}".format(type(doc\_topic)))

**print**("shape: {}".format(doc\_topic.shape))

type(doc\_topic): <type 'numpy.ndarray'>

shape: (395, 20)

Looking at the size of the output we can see that there is a distribution over the 20 topics for each of the 395 documents. These should be normalized for each document, let’s test the first 5:

**for** n **in** range(5):

sum\_pr = sum(doc\_topic[n,:])

**print**("document: {} sum: {}".format(n, sum\_pr))

document: 0 sum: 1.0

document: 1 sum: 1.0

document: 2 sum: 1.0

document: 3 sum: 1.0

document: 4 sum: 1.0

Using the title of the new stories, we can sample the most probable topic:

**for** n **in** range(10):

topic\_most\_pr = doc\_topic[n].argmax()

**print**("doc: {} topic: {}**\n**{}...".format(n,

topic\_most\_pr,

titles[n][:50]))

doc: 0 topic: 11

0 UK: Prince Charles spearheads British royal revo...

doc: 1 topic: 0

1 GERMANY: Historic Dresden church rising from WW2...

doc: 2 topic: 15

2 INDIA: Mother Teresa's condition said still unst...

doc: 3 topic: 11

3 UK: Palace warns British weekly over Charles pic...

doc: 4 topic: 15

4 INDIA: Mother Teresa, slightly stronger, blesses...

doc: 5 topic: 15

5 INDIA: Mother Teresa's condition unchanged, thou...

doc: 6 topic: 15

6 INDIA: Mother Teresa shows signs of strength, bl...

doc: 7 topic: 15

7 INDIA: Mother Teresa's condition improves, many ...

doc: 8 topic: 15

8 INDIA: Mother Teresa improves, nuns pray for "mi...

doc: 9 topic: 0

9 UK: Charles under fire over prospect of Queen Ca...

Looks pretty good except for topic 0– should docs 1 and 9 be given the same topic? Doesn’t look like it.

**Visualizing the inference**

Finally, let’s visualize some of the distributions. To do that I’m going to use matplotlib – you can see my previous posts (above) if you need help installing.

First, we import matplotlib and set a style:

**import** **matplotlib.pyplot** **as** **plt**

*# use matplotlib style sheet*

**try**:

plt.style.use('ggplot')

**except**:

*# version of matplotlib might not be recent*

**pass**

Next, let’s see what some of the topic-word distributions look like. The idea here is that each topic should have a distinct distribution of words. In the stem plots below, the height of each stem reflects the probability of the word in the focus topic:

f, ax= plt.subplots(5, 1, figsize=(8, 6), sharex=True)

**for** i, k **in** enumerate([0, 5, 9, 14, 19]):

ax[i].stem(topic\_word[k,:], linefmt='b-',

markerfmt='bo', basefmt='w-')

ax[i].set\_xlim(-50,4350)

ax[i].set\_ylim(0, 0.08)

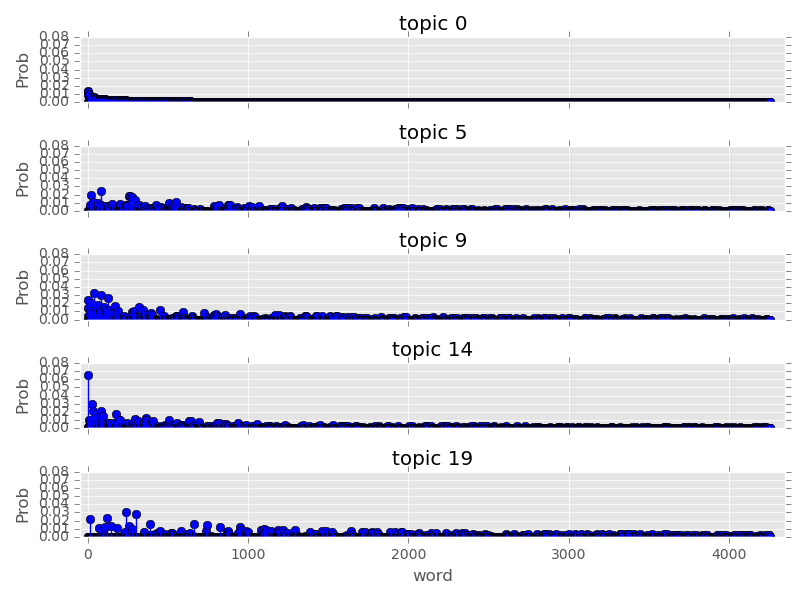
ax[i].set\_ylabel("Prob")

ax[i].set\_title("topic {}".format(k))

ax[4].set\_xlabel("word")

plt.tight\_layout()

plt.show()

**[](http://chrisstrelioff.ws/sandbox/_images/getting_started_with_latent_dirichlet_allocation_in_python_topic-work-plot_1.png)**

Finally, let’s look at the topic distribution for a few documents. These distributions give the probability of each of the 20 topics for every document. I will only plot a few:

f, ax= plt.subplots(5, 1, figsize=(8, 6), sharex=True)

**for** i, k **in** enumerate([1, 3, 4, 8, 9]):

ax[i].stem(doc\_topic[k,:], linefmt='r-',

markerfmt='ro', basefmt='w-')

ax[i].set\_xlim(-1, 21)

ax[i].set\_ylim(0, 1)

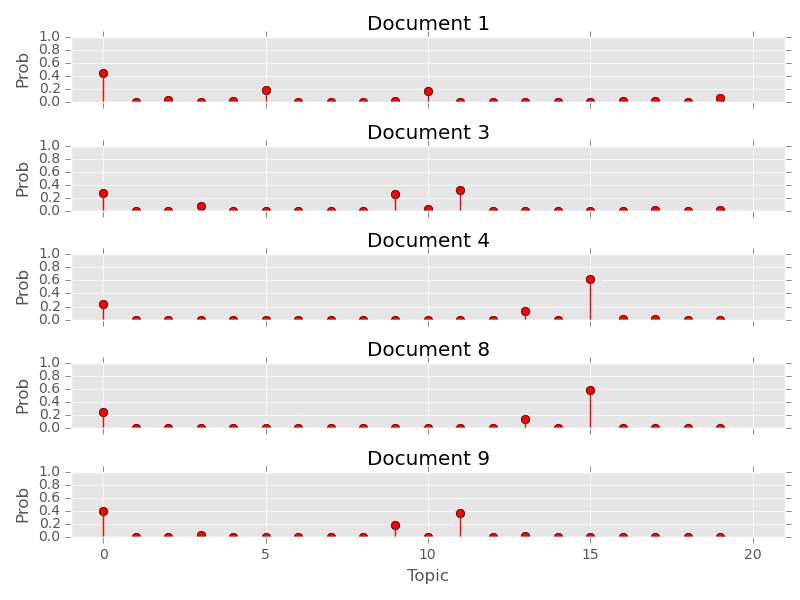
ax[i].set\_ylabel("Prob")

ax[i].set\_title("Document {}".format(k))

ax[4].set\_xlabel("Topic")

plt.tight\_layout()

plt.show()

**[](http://chrisstrelioff.ws/sandbox/_images/getting_started_with_latent_dirichlet_allocation_in_python_doc_topic-plot_1.png)**

Plotting the distribution of topics for the above documents provides an important insight: many documents have more than one topic with high probability. As a result, choosing the topic with highest probability for each document can be subject to uncertainty; *note to self: be careful*. Maybe the full distribution over topics should be considered when comparing two documents