Topic modeling in Python

This section illustrates how to do approximate topic modeling in Python. We will use a technique called non-negative matrix factorization (NMF) (https://en.wikipedia.org/wiki/Non-negative_matrix_factorization) that strongly resembles Latent Dirichlet Allocation (LDA) which we covered in the previous section, Topic modeling with MALLET (topic_model_mallet.html#topic-model-mallet). [1] Whereas LDA is a probabilistic model capable of expressing uncertainty about the placement of topics across texts and the assignment of words to topics, NMF is a deterministic algorithm which arrives at a single representation of the corpus. For this reason, NMF is often characterized as a machine learning algorithm. Like LDA, NMF arrives at its representation of a corpus in terms of something resembling "latent topics".

Note

The name "Non-negative matrix factorization" has the virtue of being transparent. A "non-negative matrix" is a matrix containing non-negative values (here zero or positive word frequencies). And factorization refers to the familiar kind of mathematical factorization. Just as a polynomial $x^2 + 3x + 2$ may be factored into a simple product $(x + 2)(x + 1)$, so too may a matrix $\begin{pmatrix} 6 & 2 & 4 \\ 9 & 3 & 6 \end{pmatrix}$ be factored into the product of two smaller matrices $\begin{pmatrix} 2 \\ 3 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}$.

This section follows the procedures described in Topic modeling with MALLET (topic_model_mallet.html#topic-model-mallet), making the substitution of NMF for LDA where appropriate.

This section uses the novels by Brontë and Austen. These novels are divided into parts as follows:

```
In [1]: import os

In [2]: CORPUS_PATH = os.path.join('data', 'austen-brontë-split')

In [3]: filenames = sorted([os.path.join(CORPUS_PATH, fn) for fn in os.listdir(CORPUS_PATH)])
```

```
# files are located in data/austen-brontë-split
In [4]: len(filenames)
Out[4]: 813

In [5]: filenames[:5]
Out[5]:
['data/austen-brontë-split/Austen_Emma0000.txt',
 'data/austen-brontë-split/Austen_Emma0001.txt',
 'data/austen-brontë-split/Austen_Emma0002.txt',
 'data/austen-brontë-split/Austen_Emma0003.txt',
 'data/austen-brontë-split/Austen_Emma0004.txt']
```

Using Non-negative matrix factorization

As always we need to give Python access to our corpus. In this case we will work with our familiar document-term matrix.

```
In [6]: import numpy as np  # a conventional alias

In [7]: import sklearn.feature_extraction.text as text

In [8]: vectorizer = text.CountVectorizer(input='filename', stop_words='english', min_df=20)

In [9]: dtm = vectorizer.fit_transform(filenames).toarray()

In [10]: vocab = np.array(vectorizer.get_feature_names())

In [11]: dtm.shape
Out[11]: (813, 2903)

In [12]: len(vocab)
Out[12]: 2903
```

By analogy with LDA, we will use NMF to get a document-topic matrix (topics here will also be referred to as “components”) and a list of top words for each topic. We will make analogy clear by using the same variable names: `doctopic` and `topic_words`

```
In [13]: from sklearn import decomposition

In [14]: num_topics = 20

In [15]: num_top_words = 20

In [16]: clf = decomposition.NMF(n_components=num_topics, random_state=1)

# this next step may take some time
```

```
doctopic = clf.fit_transform(dtm)
```

```
# print words associated with topics
In [17]: topic_words = []

In [18]: for topic in clf.components_:
   ....:     word_idx = np.argsort(topic)[::-1][0:num_top_words]
   ....:     topic_words.append([vocab[i] for i in word_idx])
   ....:
```

To make the analysis and visualization of NMF components similar to that of LDA’s topic proportions, we will scale the document-component matrix such that the component values associated with each document sum to one.

```
In [19]: doctopic = doctopic / np.sum(doctopic, axis=1, keepdims=True)
```

Now we will average those topic shares associated with the same novel together — just as we did with the topic shares from MALLET.

```
In [20]: novel_names = []

In [21]: for fn in filenames:
   ....:     basename = os.path.basename(fn)
   ....:     name, ext = os.path.splitext(basename)
   ....:     name = name.rstrip('0123456789')
   ....:     novel_names.append(name)
   ....:

# turn this into an array so we can use NumPy functions
In [22]: novel_names = np.asarray(novel_names)

In [23]: doctopic_orig = doctopic.copy()

# use method described in preprocessing section
In [24]: num_groups = len(set(novel_names))

In [25]: doctopic_grouped = np.zeros((num_groups, num_topics))

In [26]: for i, name in enumerate(sorted(set(novel_names))):
   ....:     doctopic_grouped[i, :] = np.mean(doctopic[novel_names == name, :], axis=0)
   ....:

In [27]: doctopic = doctopic_grouped
```

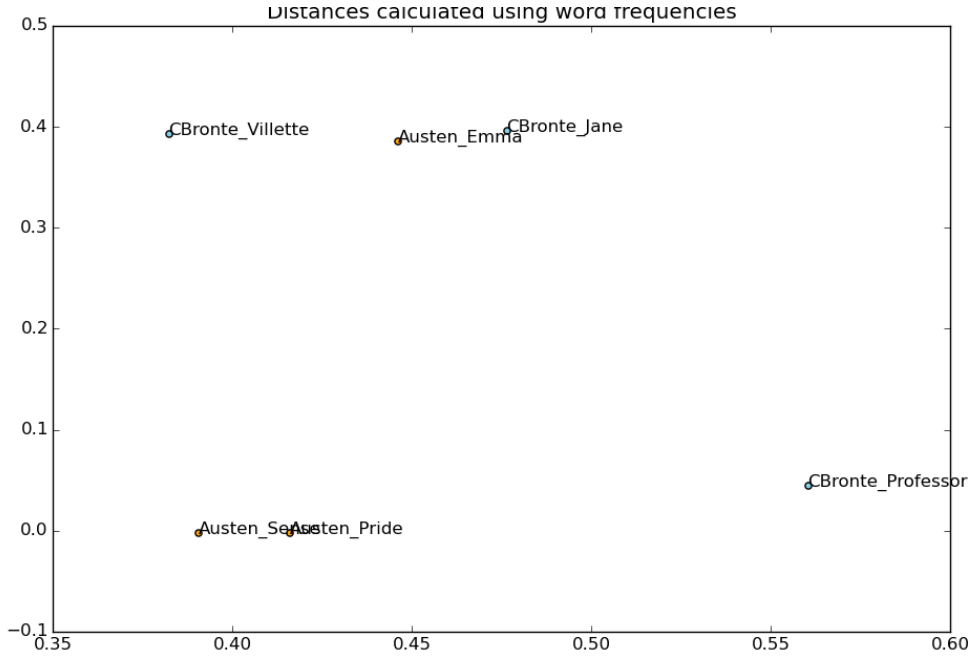
	NMF Topic 1	NMF Topic 2	NMF Topic 3	NMF Topic 4	NMF Topic 5	NMF Topic 6	NMF Topic 7	NMF Topic 8	NMF Topic 9	NMF Topic 10	NMF Topic 11	NMF Topic 12	NMF Topic 13	NMF Topic 14	NMF Topic 15
Austen_Emma	0.07	0.2	0.00	0.00	0.09	0.11	0.09	0.01	0.01	0.00	0.05	0.07	0.05	0.02	0.05

	NMF Topic 1	NMF Topic 2	NMF Topic 3	NMF Topic 4	NMF Topic 5	NMF Topic 6	NMF Topic 7	NMF Topic 8	NMF Topic 9	NMF Topic 10	NMF Topic 11	NMF Topic 12	NMF Topic 13	NMF Topic 14	NMF Topic 15
Austen_Pride	0.08	0.0	0.01	0.32	0.07	0.10	0.04	0.01	0.02	0.00	0.07	0.07	0.05	0.02	0.06
Austen_Sense	0.08	0.0	0.27	0.01	0.11	0.03	0.04	0.01	0.03	0.00	0.05	0.06	0.09	0.01	0.00
CBronte_Jane	0.10	0.0	0.00	0.00	0.04	0.08	0.06	0.02	0.08	0.00	0.06	0.04	0.06	0.10	0.07
CBronte_Professor	0.08	0.0	0.00	0.01	0.01	0.04	0.00	0.02	0.01	0.04	0.08	0.05	0.04	0.16	0.01
CBronte_Villette	0.10	0.0	0.00	0.00	0.03	0.01	0.03	0.14	0.01	0.14	0.10	0.05	0.04	0.08	0.00

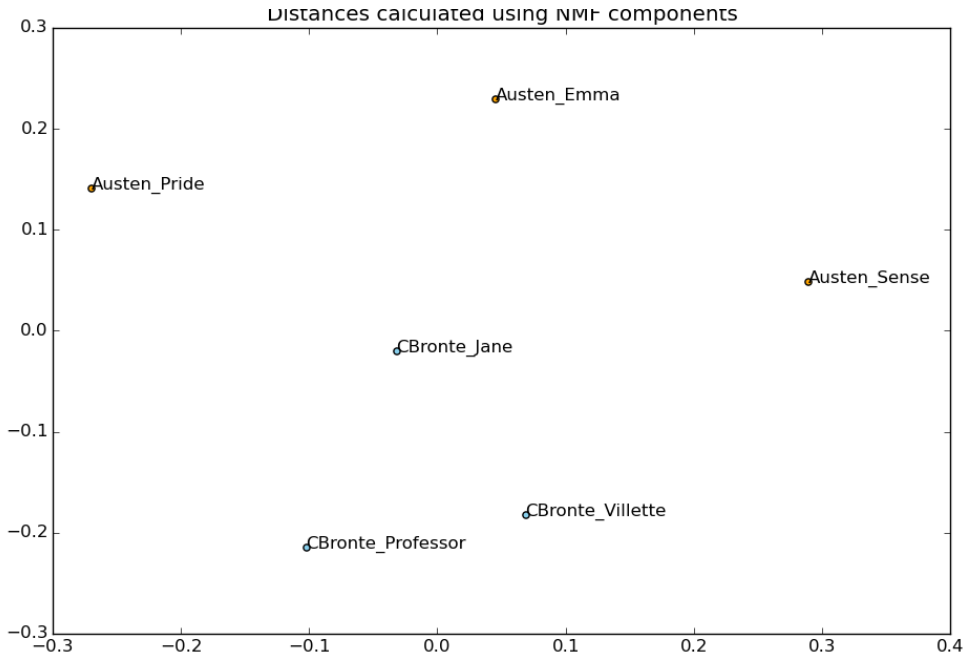
In order to fit into the space available, the table above displays the first 15 of 20 topics.

Inspecting the NMF fit

The topics (or components) of the NMF fit preserve the distances between novels (see the figures below).



(_images/plot_nmf_section_austen_brontë_cosine_mds.png)



(_images/plot_NMF_euclidean_mds.png)

Even though the NMF fit "discards" the fine-grained detail recorded in the matrix of word frequencies, the matrix factorization performed allows us to reconstruct the salient details of the underlying matrix.

As we did in the previous section, let us identify the most significant topics for each text in the corpus. This procedure does not differ in essence from the procedure for identifying the most frequent words in each text.

```
In [28]: novels = sorted(set(novel_names))

In [29]: print("Top NMF topics in...")
Top NMF topics in...

In [30]: for i in range(len(doctopic)):
   ....:     top_topics = np.argsort(doctopic[i,:])[::-1][0:3]
   ....:     top_topics_str = ' '.join(str(t) for t in top_topics)
   ....:     print("{}: {}".format(novels[i], top_topics_str))
   ....:
Austen_Emma: 1 19 5
Austen_Pride: 3 5 0
Austen_Sense: 2 17 4
CBronte_Jane: 18 16 0
CBronte_Professor: 15 13 16
CBronte_Villette: 9 7 16
```

And we already have lists of words (`topic_words`) most strongly associated with the components. For reference, we will display them again:

```
# show the top 15 words
In [31]: for t in range(len(topic_words)):
   ....:     print("Topic {}: {}".format(t, ' '.join(topic_words[t][:15])))
   ....:
Topic 0: said know think come replied tell john cried looked answer let asked yes better mean
Topic 1: emma weston knightley frank churchill mr thing little father woodhouse fairfax mrs great hartfi
eld body
Topic 2: marianne elinor willoughby sister jennings mother colonel dashwood brandon mrs time moment soon
 heart little
Topic 3: elizabeth darcy bingley bennet mr sister collins wickham soon lady family lydia catherine time
 room
Topic 4: mrs did day elton said come quite lady good time jennings party weston thing came
Topic 5: mr man knightley elton rochester think day evening great young wife thought quite time wickham
Topic 6: miss woodhouse bates quite young sure oh say fairfax great think smith temple room lady
Topic 7: little graham bretton papa dr lucy polly like home child paulina don think bassompierre thought
Topic 8: sir rochester shall yes lady don adele said john mr good say ll little long
Topic 9: madame beck dr knew little door like old good paul night english day john thought
Topic 10: did know think believe told look man say knew came tell make felt good power
Topic 11: know shall say dear oh sure thing letter think good quite yes just does like
Topic 12: john man good house father mother young years dashwood woman old family make little fortune
Topic 13: like face hunsden eyes good eye thought saw little old man look way looking crimsworth
Topic 14: jane fairfax love little oh heard think mother heart soon feel voice campbell make bates
Topic 15: monsieur little mdlle english said mademoiselle vous est pelet hand reuter frances pupils henr
i time
Topic 16: day time long night thought life heart hour felt saw hand morning mind left evening
Topic 17: edward elinor lucy ferrars dashwood think time sister say mother sure thing soon brother make
Topic 18: room door rochester house heard night little looked stood came hall come bessie did long
Topic 19: harriet emma elton mr friend knightley martin woodhouse think good thing little smith thought
 mind
```

There are many ways to inspect and to visualize topic models. Some of the most common methods are covered in Visualizing topic models (topic_model_visualization.html#topic-model-visualization).

Distinctive topics

Consider the task of finding the topics that are distinctive of Austen using the NMF “topics”. Using the simple difference-in-averages we can find topics that to be associated with Austen’s novels rather than Brontë’s.

```
In [32]: austen_indices, cbronte_indices = [], []

In [33]: for index, fn in enumerate(sorted(set(novel_names))):
   ....:     if "Austen" in fn:
   ....:         austen_indices.append(index)
   ....:     elif "CBronte" in fn:
   ....:         cbronte_indices.append(index)
   ....:

In [34]: austen_avg = np.mean(doctopic[austen_indices, :], axis=0)

In [35]: cbronte_avg = np.mean(doctopic[cbronte_indices, :], axis=0)

In [36]: keyness = np.abs(austen_avg - cbronte_avg)

In [37]: ranking = np.argsort(keyness)[::-1]  # from highest to lowest; [::-1] reverses order in Python
 sequences

# distinctive topics:
In [38]: ranking[:10]
Out[38]: array([ 3, 15, 13,  2, 16, 18,  1,  4,  9,  7])
```

	Topic 1	**Topic 2**	**Topic 3**	**Topic 4**	**Topic 7**	**Topic 9**	**Topic 13**	**Topic 15**	**Topic 16**	**Topic 18**
Austen_Emma	0.2	0.00	0.00	0.09	0.01	0.00	0.02	0.01	0.03	0.01
Austen_Pride	0.0	0.01	0.32	0.07	0.01	0.00	0.02	0.01	0.04	0.02
Austen_Sense	0.0	0.27	0.01	0.11	0.01	0.00	0.01	0.01	0.04	0.02
CBronte_Jane	0.0	0.00	0.00	0.04	0.02	0.00	0.10	0.02	0.11	0.14
CBronte_Professor	0.0	0.00	0.01	0.01	0.02	0.04	0.16	0.23	0.12	0.09
CBronte_Villette	0.0	0.00	0.00	0.03	0.14	0.14	0.08	0.08	0.13	0.06

Topic 1	emma	weston	knightley	frank	churchill	mr	thing	little	father	woodhouse
Topic 2	marianne	elinor	willoughby	sister	jennings	mother	colonel	dashwood	brandon	mrs
Topic 3	elizabeth	darcy	bingley	bennet	mr	sister	collins	wickham	soon	lady
Topic 4	mrs	did	day	elton	said	come	quite	lady	good	time
Topic 7	little	graham	bretton	papa	dr	lucy	polly	like	home	child
Topic 9	madame	beck	dr	knew	little	door	like	old	good	paul
Topic 13	like	face	hunsden	eyes	good	eye	thought	saw	little	old
Topic 15	monsieur	little	mdlle	english	said	mademoiselle	vous	est	pelet	hand
Topic 16	day	time	long	night	thought	life	heart	hour	felt	saw
Topic 18	room	door	rochester	house	heard	night	little	looked	stood	came

[1] While there are significant differences between NMF and LDA, there are also similarities. Indeed, if the texts in a corpus have certain properties, NMF and LDA will arrive at the same representation of a corpus [AGH+13] (references.html#arora-practical-2013).