hw5_kleislemurphy

May 11, 2021

1 HW5: Topic Models and LDA

STATS271/371: Applied Bayesian Statistics

Stanford University. Spring, 2021.

Name: Isaac Kleisle-Murphy

Names of any collaborators: Anmol, Yan

Due: 11:59pm Monday, May 10, 2021 via GradeScope

Recall the following generatize model for LDA. Suppose we have K topics and N documents.

For each topic $k \leq K$, draw a topic

$$\eta_k \sim \text{Dir}(\phi)$$

Then, for each document $n \leq N$, draw topic proportions

$$\pi_n \sim \text{Dir}(\alpha)$$

Finally, for each word l in document n, first draw a topic assignment

$$z_{n,l} \mid \pi_n \sim \text{Cat}(\pi_n)$$

and draw a word

$$x_{n,l} \mid z_{n,l} \sim \text{Cat}(z_{n,l})$$

As mentioned in class, while this formulation is easier to present, it's more efficient to represent the documents as sparse vectors of *word counts*, $\mathbf{y}_n \in \mathbb{N}^V$ where $y_{n,v} = \sum_{l=1}^{L} \mathbb{I}[x_{n,l} = v]$.

In this assignment, we will be re-exploring the Federalist papers in their entirety. We've provided a $N \times V$ dataframe of the essays represented as word counts. The rows of the data frame correspond to the 85 individual essays and the columns correspond to the 5320 words in the vocabulary. We have already preprocessed the raw essays to remove very common and very infrequent words.

Using this data, we will fit a topic model and do some analysis.

1.1 Problem 1: Fit LDA on this data set.

Fit a 10 topic LDA on the data using CAVI. For each topic, output the top 5 words. You might find the structure in the Poisson matrix factorization notebook helpful. (Note that that notebook used the JAX backend available in the `tfp-nightly` package, but you could have used the regular TensorFlow Probability package instead. The nice thing about TFP is that its functions broadcast nicely, which is helpful when we have lots of factors in the mean field variational posterior.

First, a touch of setup:

[2]:
```
!pip install tfp-nightly
```

```
Collecting tfp-nightly
  Downloading https://files.pythonhosted.org/packages/35/69/0a2acb047c218f
2a999bdc0fe60e20d8ef9950a65fe5a6f20905028d69e5/tfp_nightly-0.13.0.dev20210511-py
2.py3-none-any.whl (5.4MB)
     |                  | 5.4MB 10.2MB/s
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (0.1.6)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (4.4.2)
Requirement already satisfied: cloudpickle>=1.3 in
/usr/local/lib/python3.7/dist-packages (from tfp-nightly) (1.3.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (1.19.5)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (1.15.0)
Requirement already satisfied: gast>=0.3.2 in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (0.3.3)
Installing collected packages: tfp-nightly
Successfully installed tfp-nightly-0.13.0.dev20210511
```

[3]:
```
import numpy as onp
import pandas as pd
import matplotlib.pyplot as plt
from tqdm.auto import trange

from jax import jit
import jax.numpy as np
import jax.scipy.special as spsp
import jax.random as jr

import tensorflow_probability as tfp
import tensorflow_probability.substrates.jax.distributions as tfd

from scipy.stats import entropy

from tqdm import tqdm
```

```
import matplotlib.pyplot as plt

tokenized_fed = pd.read_csv('https://raw.githubusercontent.com/slinderman/
 ↪stats271sp2021/main/assignments/hw5/tokenized_fed.csv').iloc[:, 1:]
authorship = pd.read_csv('https://raw.githubusercontent.com/slinderman/
 ↪stats271sp2021/main/assignments/hw5/authorship.csv')
print('Shape (tokenized_fed): ', tokenized_fed.shape)
print('Shape (authorship): ', authorship.shape)

tokenized_fed.head()
```

```
Shape (tokenized_fed):  (85, 5320)
Shape (authorship):  (85, 2)
```

[3]:

	unequivocal	experience	inefficacy	…	habeas	corpus	clerks
0	1.0	1.0	1.0	…	0.0	0.0	0.0
1	0.0	2.0	0.0	…	0.0	0.0	0.0
2	0.0	1.0	0.0	…	0.0	0.0	0.0
3	0.0	2.0	0.0	…	0.0	0.0	0.0
4	0.0	1.0	0.0	…	0.0	0.0	0.0

[5 rows x 5320 columns]

Next, we define our CAVI functions.

[4]:
```
def eqlog(dirch):
  t1=spsp.digamma(dirch.concentration)
  t2 = spsp.digamma(
      dirch.concentration @ np.ones((dirch.concentration.shape[1],1))
  )
  return t1 - t2

# def eqlog_(dirch):
#   return np.add(
#     spsp.digamma(dirch.concentration),
#     - np.vstack([spsp.digamma(dirch.concentration.sum(axis=1))
#                  for k in range(dirch.concentration.shape[1])]).T
#   )

def cavi_update_z(q_pi, q_eta, N, K, V, data):

  pi_exp_log = eqlog(q_pi).reshape(N, 1, K)
  eta_exp_log = eqlog(q_eta).reshape(1, V, K)
  logits_lambda_z = pi_exp_log + eta_exp_log

  return tfd.Multinomial(data, logits=logits_lambda_z)
```

```
def update_cavi(qtuple, phi_pi, phi_eta, **kwargs):

  # unpack
  q_pi, q_eta, q_z = qtuple

  # update expectations
  # note mean is probability multiplied by observed counts, so we're good
  E_log_pi = q_z.mean().sum(axis=1) # np.stack([q_z.mean()[:, :, k].sum(axis=1)␣
 ↪for k in range(K)])
  E_log_eta = q_z.mean().sum(axis=0).T # np.stack([q_z.mean()[:, :, k].
 ↪sum(axis=0) for k in range(K)])

  q_z = cavi_update_z(q_pi, q_eta, **kwargs)
  q_pi = tfd.Dirichlet(phi_pi + E_log_pi) # alphas included here
  q_eta = tfd.Dirichlet(phi_eta + E_log_eta) # alphas also included here

  return q_pi, q_eta, q_z

def elbo(q_pi, q_eta, q_z, phi_pi, phi_eta, N, K, V):

  # the "first two" terms: prior
  first_two_terms = q_pi.cross_entropy(tfd.Dirichlet(phi_pi)).sum() +\
    q_eta.cross_entropy(tfd.Dirichlet(phi_eta)).sum()
  # last three terms
  last_three_terms = -(
      q_pi.entropy().sum() + \
        q_eta.entropy().sum() + \
        np.nansum(entropy(q_z.mean(), axis=2)) #q_z.entropy().sum()
  )
  # middle terms
  middle_term_one = np.sum((q_z.mean() * (eqlog(q_pi).reshape(N, 1, K))))
  middle_term_two = np.sum((q_z.mean() * (eqlog(q_eta).T.reshape(1, V, K))))

  return -first_two_terms + middle_term_one + middle_term_two - last_three_terms

def cavi(data, K=10, phi_pi=None, phi_eta=None, n_iter=1000):
  data = data.astype(np.float32)
  N, V = data.shape
  if phi_pi is None:
    phi_pi = np.ones((N, K))
  if phi_eta is None:
    phi_eta = np.ones((K, V))

  q_pi = tfd.Dirichlet(phi_pi)
  q_eta = tfd.Dirichlet(phi_eta) # across is each distribution
  q_z = tfd.Multinomial(data, logits=np.zeros((N, V, K)))
```

```
  elbos = []
  for i in trange(n_iter):
    q_pi, q_eta, q_z = update_cavi((q_pi, q_eta, q_z), phi_pi, phi_eta,
                                   N=N, K=K, V=V, data=data)
    elbos.append(elbo(q_pi, q_eta, q_z, phi_pi, phi_eta, N, K, V))
  return elbos, q_pi, q_eta, q_z
```

Now, we're ready to go. We'll sample our CAVI in the following manner: we'll use $\alpha = 1$ priors on both Dirichlet distributions. Further, we'll initialize the multinomial z's to uniform probability, and the π, η Dirichlets to the prior $\alpha_\pi = 1 = \alpha_\eta$. Of course, these initializations settings would be tune-worthy (i.e. on a heldout set) in a larger scale project. We'll go for 1,000 iterations here.

[5]:
```
elbo_out, q_pi, q_eta, q_z = cavi(data=tokenized_fed.values, K=10, n_iter=1000)
```

```
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0
and rerun for more info.)
```

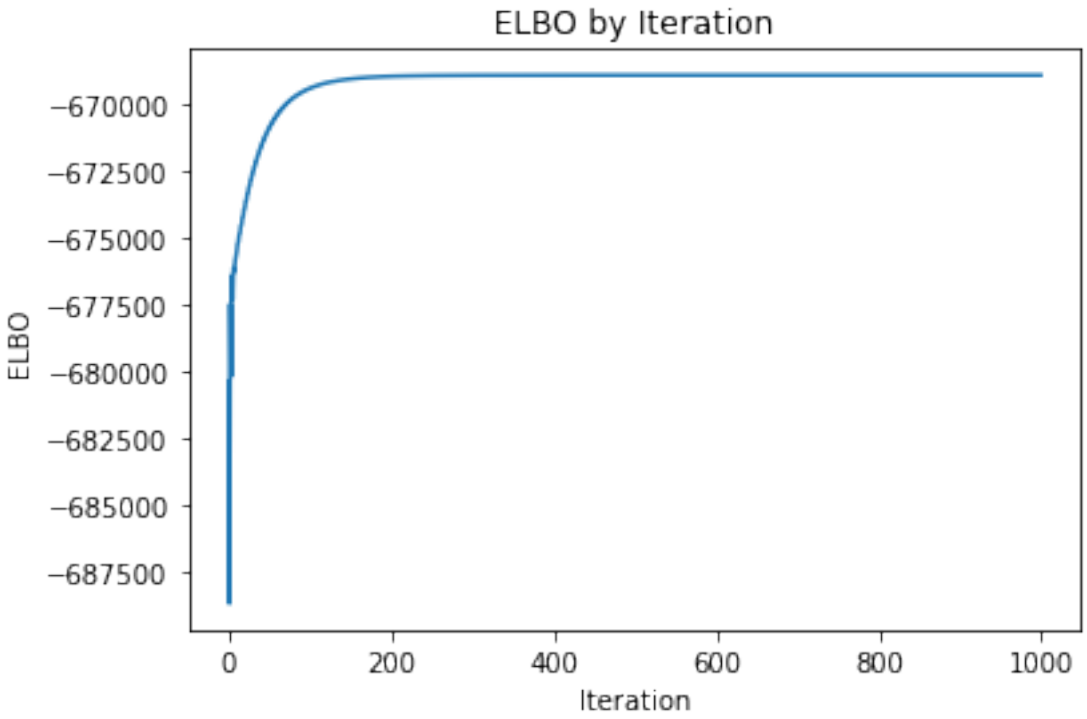
```
HBox(children=(FloatProgress(value=0.0, max=1000.0), HTML(value='')))
```

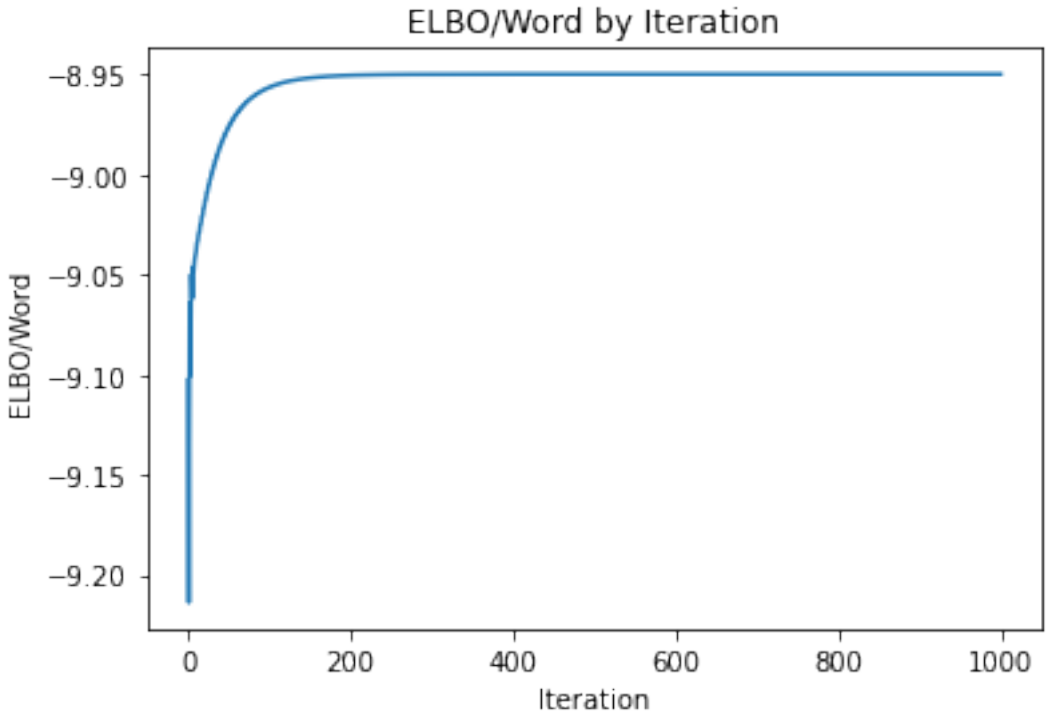
```
/usr/local/lib/python3.7/dist-
packages/scipy/stats/_distn_infrastructure.py:2664: RuntimeWarning: invalid
value encountered in true_divide
  pk = 1.0*pk / np.sum(pk, axis=axis, keepdims=True)
```

We can view the convergence, both on a total ELBO and ELBO/word scale. Clearly, our CAVI implementation has converged.

[8]:
```
plt.plot(elbo_out)
plt.title('ELBO by Iteration')
plt.xlabel('Iteration')
plt.ylabel('ELBO')
plt.show()

plt.plot([item/np.sum(tokenized_fed.values) for item in elbo_out])
plt.title('ELBO/Word by Iteration')
plt.xlabel('Iteration')
plt.ylabel('ELBO/Word')
plt.show()
```


ELBO by Iteration
ELBO
−670000
−672500
−675000
−677500
−680000
−682500
−685000
−687500
0
200
400
600
800
1000
Iteration


ELBO/Word by Iteration
ELBO/Word
−8.95
−9.00
−9.05
−9.10
−9.15
−9.20
0
200
400
600
800
1000
Iteration

Furthermore, the top words from each topic are generic and plausible. We have:

[9]:
```
top_words = pd.DataFrame(q_eta.mean().T,
                         columns=[f"topic_{item}" for item in range(1, 11)])
top_words['word'] = tokenized_fed.columns.to_list()

for i in range(1, 11):
  print("*" * 20)
  print(f"Topic {i}")
  word_head = top_words.\
    sort_values(f"topic_{i}", ascending=False)[['word', f"topic_{i}"]].\
      head().\
      reset_index(drop=True)
  print(word_head)
```

```
********************
Topic 1
           word   topic_1
0           new  0.011124
1          plan  0.007521
2           men  0.006814
3  constitution  0.004587
4           man  0.003920
********************
Topic 2
         word   topic_2
0     federal  0.015487
1        part  0.008293
2  particular  0.006832
3     without  0.005812
4     nations  0.005040
********************
Topic 3
          word   topic_3
0  legislature  0.007394
1        great  0.006841
2        force  0.006089
3       little  0.005477
4      general  0.005414
********************
Topic 4
       word   topic_4
0    people  0.013723
1        us  0.006228
2  national  0.005886
3     among  0.005066
4     first  0.004235
********************
```

```
Topic 5
           word   topic_5
0      congress  0.004567
1     authority  0.003757
2  constitution  0.003528
3    convention  0.003460
4        either  0.003260
********************
Topic 6
      word   topic_6
0  members  0.006382
1      two  0.004826
2     part  0.003665
3  federal  0.003605
4     well  0.003207
********************
Topic 7
       word   topic_7
0     union  0.008038
1    united  0.007823
2    public  0.006965
3  interest  0.005601
4  national  0.005314
********************
Topic 8
              word   topic_8
0           people  0.010118
1  representatives  0.006960
2     constitution  0.004713
3             long  0.003851
4        different  0.003815
********************
Topic 9
       word   topic_9
0      well  0.007497
1  national  0.003843
2      made  0.003703
3     great  0.003359
4    judges  0.002694
********************
Topic 10
           word  topic_10
0  constitution  0.013727
1         union  0.006110
2     necessary  0.005275
3      treaties  0.003777
4        public  0.003377
```

1.2 Problem 2: Analysis/Exploration

Using the model, for each essay assign it the most likely topic. For the undisputed papers, plot the histogram of this topic usage vs author.

[55]:
```
#load authorship
authorship = pd.read_csv('https://raw.githubusercontent.com/slinderman/
 ↪stats271sp2021/main/assignments/hw5/authorship.csv')
authorship.columns = ['text', 'author']
authorship.head()
```

[55]:

	text	author
0	1	HAMILTON
1	2	JAY
2	3	JAY
3	4	JAY
4	5	JAY

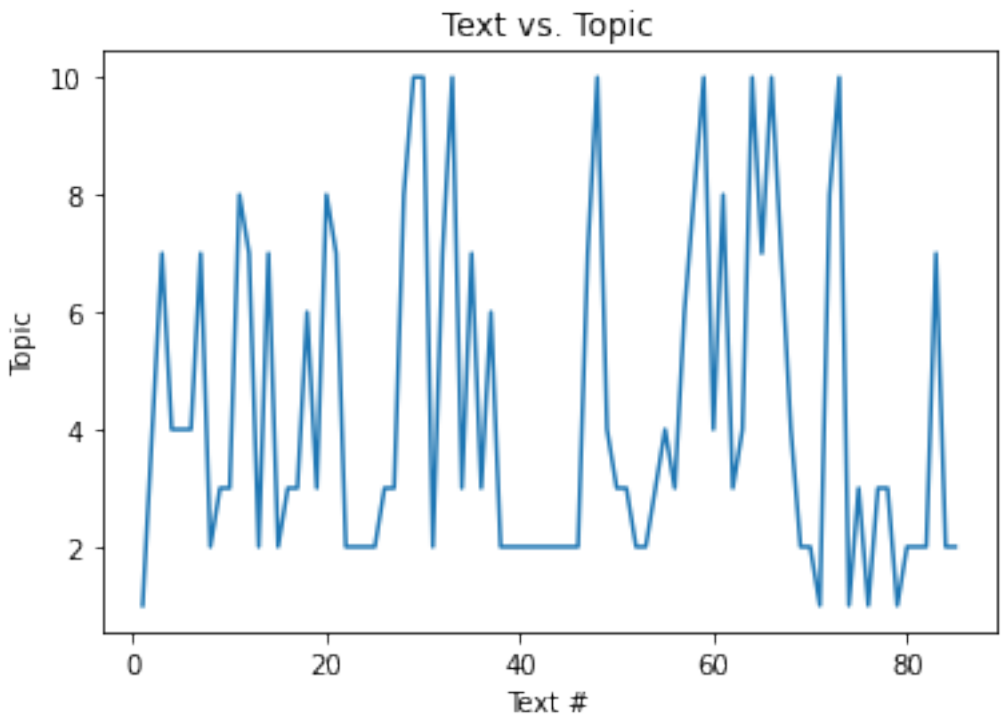
First we note the most likely topics from each paper.

[25]:
```
winners = []
data = tokenized_fed.values
for n in trange(data.shape[0]):
  winners.append(int(np.argmax(q_pi.mean()[n, :])) + 1)

authorship['probable_topic'] = winners
plt.plot(authorship.text, authorship.probable_topic)
plt.xlabel('Text #')
plt.ylabel('Topic')
plt.title('Text vs. Topic')
plt.show()

authorship
```

```
HBox(children=(FloatProgress(value=0.0, max=85.0), HTML(value='')))
```



[25]:

	text	author	probable_topic
0	1	HAMILTON	1
1	2	JAY	4
2	3	JAY	7
3	4	JAY	4
4	5	JAY	4
..	…	…	…
80	81	HAMILTON	2
81	82	HAMILTON	2
82	83	HAMILTON	7
83	84	HAMILTON	2
84	85	HAMILTON	2

[85 rows x 3 columns]

Next, we make the requested histogram of each author's probable topic distribution.

[56]:
```
authorship_ = authorship.copy()
authorship_ = pd.concat([
  authorship_,
  pd.DataFrame(q_pi.mean(), columns=[f'p_topic_{i}' for i in range(1, 11)])
  ],
  axis=1
)
```

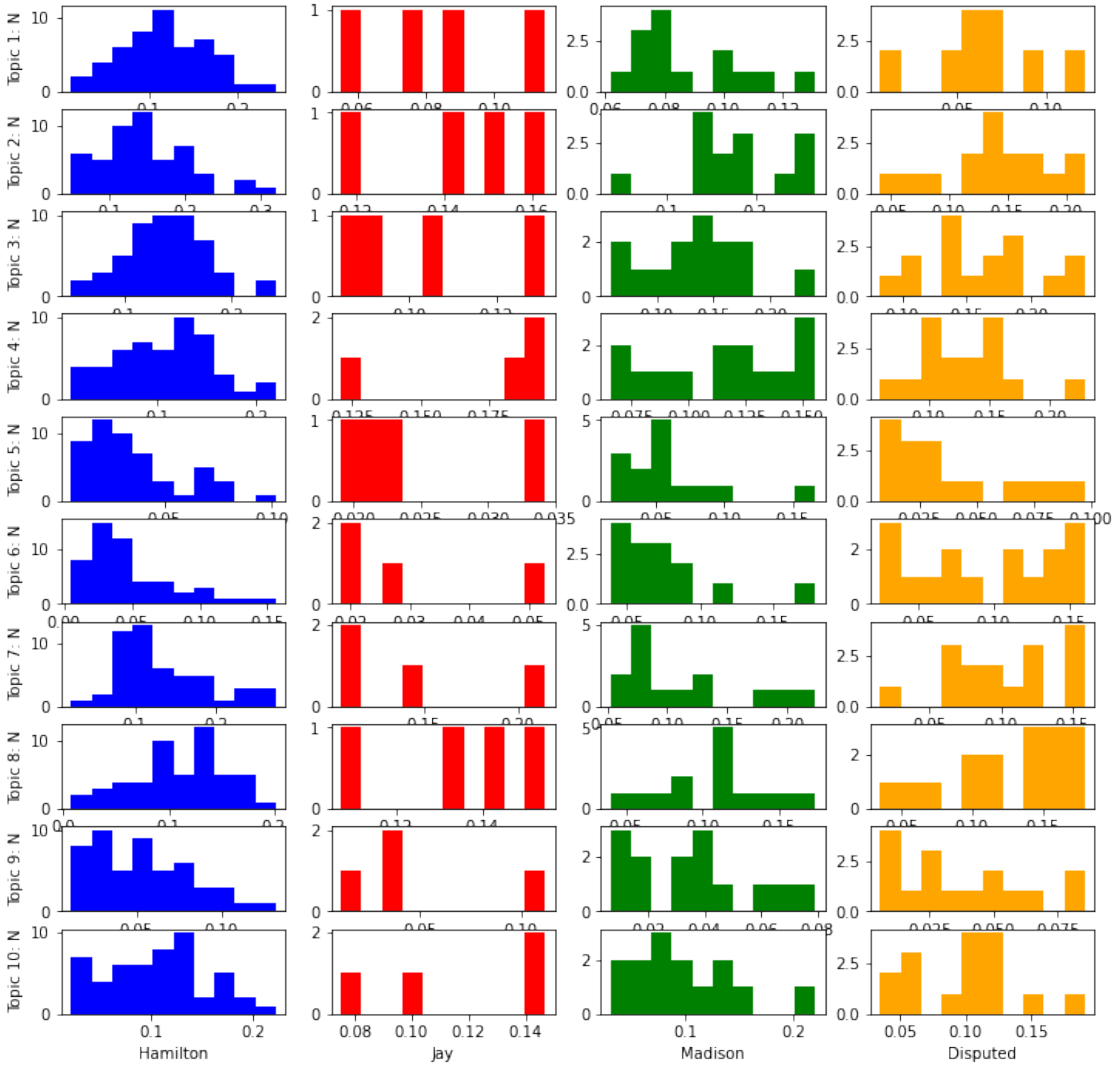
```
authorship_
```

[56]:

	text	author	p_topic_1	…	p_topic_8	p_topic_9	p_topic_10
0	1	HAMILTON	0.173961	…	0.087400	0.079595	0.128295
1	2	JAY	0.088400	…	0.134093	0.037279	0.146894
2	3	JAY	0.055136	…	0.107098	0.112163	0.075251
3	4	JAY	0.115070	…	0.154703	0.034443	0.096846
4	5	JAY	0.076223	…	0.141648	0.010926	0.145049
..	…	…	…	…	…	…	…
80	81	HAMILTON	0.186148	…	0.035445	0.037344	0.068106
81	82	HAMILTON	0.145375	…	0.007327	0.032366	0.095179
82	83	HAMILTON	0.148837	…	0.129433	0.025200	0.049937
83	84	HAMILTON	0.116501	…	0.130352	0.055502	0.097158
84	85	HAMILTON	0.152131	…	0.125166	0.081267	0.119302

[85 rows x 12 columns]

[58]:
```
fig, axs = plt.subplots(10, 4, figsize=(12, 12))
for topic in range(1,11):
  axs[topic-1, 0].hist(authorship_.query('author ==␣
 ↪"HAMILTON"')[f"p_topic_{topic}"], color='blue')
  axs[topic-1, 0].set_ylabel(f"Topic {topic}: N")
  axs[topic-1, 1].hist(authorship_.query('author ==␣
 ↪"JAY"')[f"p_topic_{topic}"], color='red')
  axs[topic-1, 2].hist(authorship_.query('author ==␣
 ↪"MADISON"')[f"p_topic_{topic}"], color='green')
  axs[topic-1, 3].hist(authorship_.query('author ==␣
 ↪"DISPUTED"')[f"p_topic_{topic}"], color='orange')
axs[9, 0].set_xlabel('Hamilton')
axs[9, 1].set_xlabel('Jay')
axs[9, 2].set_xlabel('Madison')
axs[9, 3].set_xlabel('Disputed')
plt.show()
```



1.3 Problem 3: Short Answer questions

1.3.1 Part a)

Explain what approach you would take if you wanted to use LDA to help settle disputed authorship. How would you incorporate authorship by different authors into your model?

Two immediate options pop to mind. The first, and most simplistic, involves letting each author be their own "topic." That is, there would be $K = 3$, one for Jay/Hamilton/Madison, and the assumption would be that each author would have their own political/governmental/constitutional interests, and their collective writing style about those interests would manifest itself as an all-encompassing "topic." Just as the "sports" section of a newspaper involves words relating to sports, or the "business" section involves more financial terminology, the "Hamilton" section would feature Hamilton's diction and interests, etc.

Of course, by allowing the authors to hijack the topics, the model might also struggle to account for literal topics themselves – i.e. it would distinguish (or at least attempt to) authors, but not the sub-categories they would write about. The above method would have no way of knowing that regardless of author, writing about judicial matters may increase the likelihood of certain words (i.e. court, justice, trial, bench, etc.), whereas writing about legislative matters may increase the likelihood of other words (congress, house, representative, etc.). In this way, the above model would fail to consider the fact that some subject matters – regardless of author – invoke certain terminology. To correct for this, in the model, one might want to add a second dimension to the z variable, in effect making it a tuple of $\langle$author, political topic $\rangle$. Naturally, two ηs and two πs – one for author and one for political topic – would be necessary as well. In this way, the model could view word choice as the product of both author and subject matter – and their interaction – and perhaps address the disputed texts. ### Part b)

A shortcoming of LDA discussed in this class is the fact that the model is exchangeable (which is not a very reasonable assumption for essays). What would you do to address this shortcoming? In essence, how could you account for dependencies between words that are near each other in the essay?

One obvious issue is that this exchangeable structure fails to account for the ordering of words, and potential stylistic tendencies therein. For instance, Hamilton might be inclined to write something like (Federalist 83):

"The force of this consideration is, however, diminished by others. The sheriff, who is the summoner of ordinary

But Madison might have written:

"However, the force of this consideration is diminished by others. The sheriff, who is the summoner of ordinary

For the purposes of the model, this reversal (and more generally, the reversal of clauses) would be the same to the model. As such, it would lose important stylistic information.

I see a couple of potential remedies here. First, we could concatenate phrases (i.e. "ordinary" and "juries" into "ordinary_juries", and treat them as their own word when appropriate. Similarly, we could treat opening-sentence "however"'s as different than mid-sentence "however"'s. In this way, phrasing might better be captured by our LDA model.

Alternatively, if we were really insistent on phrasing/ordering, we could scrap the LDA model and move to a Markov-chain approach. Each of the topics could still have its own "sub-chain" (i.e. square region of the transition matrix, with some positive probability of transitioning out to another topic), and we could proceed this way. This, however, might be much less computationally convenient than the LDA set forth above.

2 Submission Instructions

Formatting: check that your code does not exceed 80 characters in line width. If you're working in Colab, you can set *Tools → Settings → Editor → Vertical ruler column* to 80 to see when you've exceeded the limit.

Download your notebook in .ipynb format and use the following commands to convert it to PDF:

```
jupyter nbconvert --to pdf hw5_yourname.ipynb
```

Dependencies:

- `nbconvert`: If you're using Anaconda for package management,

```
conda install -c anaconda nbconvert
```

Upload your .ipynb and .pdf files to Gradescope.