hw5 (2)

May 10, 2021

1 HW5: Topic Models and LDA

STATS271/371: Applied Bayesian Statistics

Stanford University. Spring, 2021.

Name: Jake Taylor

Names of any collaborators: *Names here*

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.

[ ]:
```
import pandas as pd

# Load the data
data = pd.read_csv('tokenized_fed.csv', index_col = 0)
data
```

[ ]:

	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
..	…	…	…	…	…	…	…
80	0.0	0.0	0.0	…	0.0	0.0	0.0
81	0.0	0.0	0.0	…	0.0	0.0	0.0
82	0.0	1.0	0.0	…	1.0	1.0	1.0
83	0.0	0.0	0.0	…	3.0	3.0	1.0
84	0.0	2.0	0.0	…	0.0	0.0	0.0

[85 rows x 5320 columns]

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.

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

```
Collecting tfp-nightly
  Downloading https://files.pythonhosted.org/packages/2a/c4/ff866e70fa4a15
6f85c70d14622e111e4a84bdd295b932d88b6e041745ea/tfp_nightly-0.13.0.dev20210510-py
2.py3-none-any.whl (5.4MB)
     |                  | 5.4MB 2.8MB/s
Requirement already satisfied: cloudpickle>=1.3 in
/usr/local/lib/python3.7/dist-packages (from tfp-nightly) (1.3.0)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages
(from tfp-nightly) (0.1.6)
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)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from tfp-nightly) (4.4.2)
Installing collected packages: tfp-nightly
Successfully installed tfp-nightly-0.13.0.dev20210510
```

[ ]:
```
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
```

[ ]:
```
y_nv = data.to_numpy()
```

[ ]:
```
class LDA:
  """Implmentation of LDA."""
  def __init__(self, data, K=10, alpha=1, phi=1):
```

```
    assert isinstance(data, np.ndarray)
    self.y_nv = data.astype(np.float32)
    self.N, self.V = self.y_nv.shape
    self.K = K
    self.alpha = alpha
    self.phi = phi

    onp.random.seed(123)
    self.q_pi = tfd.Dirichlet(self.alpha * onp.random.rand(self.N, self.K).
↪astype(np.float32))
    self.q_eta = tfd.Dirichlet(self.phi * onp.random.rand(self.K, self.V).
↪astype(np.float32))

    self.q_z = tfd.Multinomial(self.y_nv, logits=onp.zeros((self.N, self.V,␣
↪self.K)).astype(np.float32))

    self.q_pi_prior = tfd.Dirichlet(self.alpha * onp.ones((self.N, self.K)).
↪astype(np.float32))
    self.q_eta_prior = tfd.Dirichlet(self.phi * onp.ones((self.K, self.V)).
↪astype(np.float32))

    self.E_z = self.q_z.mean()
    self.E_log_pi = self.expected_log_dirichlet(self.q_pi)
    self.E_log_eta = self.expected_log_dirichlet(self.q_eta)
    self.log_lambda_z_nvk = self.E_log_pi[:, None, :] + self.E_log_eta.T

  @staticmethod
  def expected_log_dirichlet(dirichlet):
    """Helper function."""
    alpha = dirichlet.concentration
    return spsp.digamma(alpha) - spsp.digamma(alpha.sum(axis=1)[...,None])

  def cavi_step(self):
    """One step of CAVI."""
    self.E_log_pi = self.expected_log_dirichlet(self.q_pi)
    self.E_log_eta = self.expected_log_dirichlet(self.q_eta)
    assert self.E_log_pi.shape == (self.N, self.K)
    assert self.E_log_eta.shape == (self.K, self.V)

    self.log_lambda_z_nvk = self.E_log_pi[:, None, :] + self.E_log_eta.T
    assert self.log_lambda_z_nvk.shape == (self.N, self.V, self.K)

    self.q_z = tfd.Multinomial(self.y_nv, logits=self.log_lambda_z_nvk,␣
↪name='topic_assignments')
    self.E_z = self.q_z.mean()
    assert self.E_z.shape == (self.N, self.V, self.K)
```

```
    self.E_N_nk = self.E_z.sum(axis=1)
    self.E_N_kv = self.E_z.sum(axis=0).T
    assert self.E_N_nk.shape == (self.N, self.K)
    assert self.E_N_kv.shape == (self.K, self.V)

    self.q_pi = tfd.Dirichlet(self.alpha + self.E_N_nk,␣
↪name='topic_proportions')
    self.q_eta = tfd.Dirichlet(self.phi + self.E_N_kv, name='topic_parameters')

  def elbo(self, array_return=False):
    """Compute the evidence lower bound."""
    elbo = []
    elbo.append(-self.q_pi.kl_divergence(self.q_pi_prior).sum())
    elbo.append(-self.q_eta.kl_divergence(self.q_eta_prior).sum())
    elbo.append(np.multiply(self.E_log_pi[:, None, :], self.E_z).sum())
    elbo.append(np.multiply(self.E_log_eta.T[None, :, :], self.E_z).sum())
    elbo.append(-np.multiply(np.log(self.q_z.probs_parameter()), self.E_z).
↪sum())
    if array_return:
      return np.array(elbo)
    return np.array(elbo).sum() / (self.y_nv.sum())

  def cavi(self, num_iters=100):
    """Run coordinate ascent VI for LDA."""
    elbos = [self.elbo()]
    for itr in trange(num_iters):
        self.cavi_step()
        elbos.append(self.elbo())

    return np.array(elbos), (self.q_z, self.q_pi, self.q_eta)
```

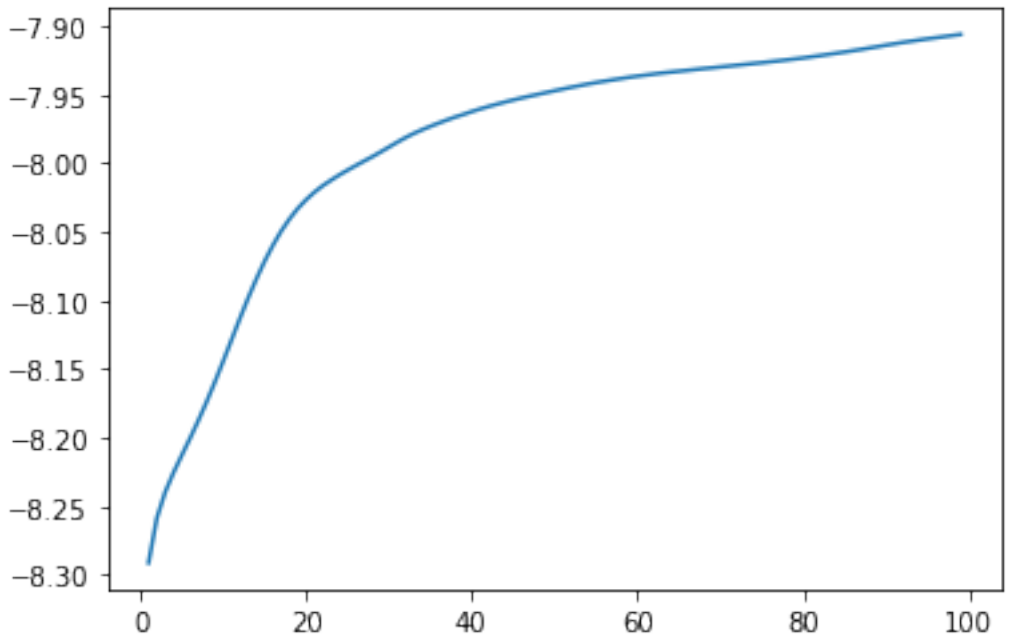
[411]:
```
test = LDA(y_nv, alpha=1, phi=1)
test.elbo(array_return=True)
```

[411]:
```
DeviceArray([  -15573.701,  -670138.9  , -1445088.   , -1358432.2  ,
               172065.84 ], dtype=float32)
```

[412]:
```
res = test.cavi(num_iters=100)
```

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

[413]:
```
plt.plot(res[0][1:])
plt.show()
```



[414]:
```
top5 = res[1][2].mean().argsort(axis=1)[...,-5:].copy()
```

[415]:
```
data.columns[top5]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and
will be removed in a future version.  Convert to a numpy array before indexing
instead.
  """Entry point for launching an IPython kernel.
```

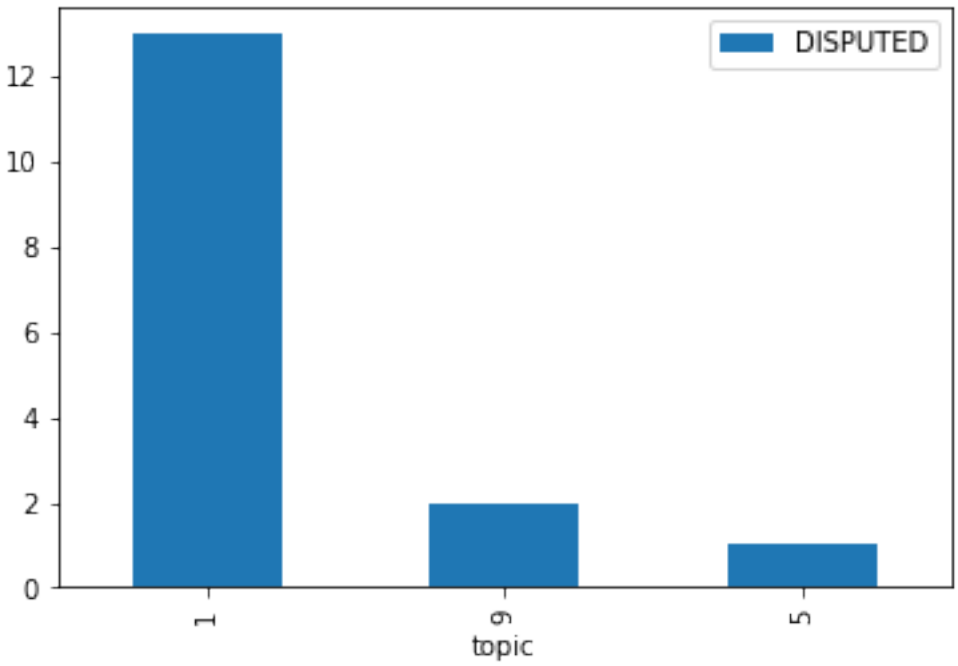
[415]:
```
array([['distant', 'tranquillity', 'direction', 'separation', 'northern'],
       ['federal', 'national', 'union', 'constitution', 'people'],
       ['biennial', 'representatives', 'electors', 'elections', 'slaves'],
       ['distant', 'tranquillity', 'direction', 'separation', 'northern'],
       ['departments', 'judiciary', 'department', 'legislative',
        'executive'],
       ['league', 'greece', 'achaeans', 'macedon', 'cities'],
       ['man', 'body', 'senate', 'president', 'executive'],
       ['ships', 'navy', 'navigation', 'markets', 'trade'],
       ['jurisdiction', 'court', 'jury', 'cases', 'courts'],
       ['authority', 'foreign', 'peace', 'war', 'nations']], dtype=object)
```

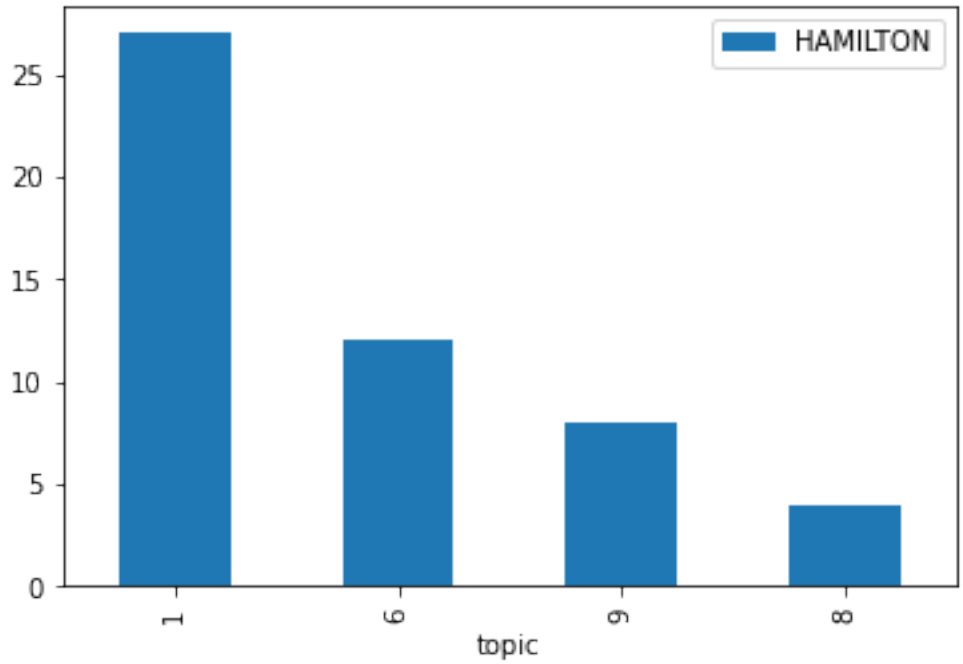
1.2 Problem 2: Analysis/Exploration

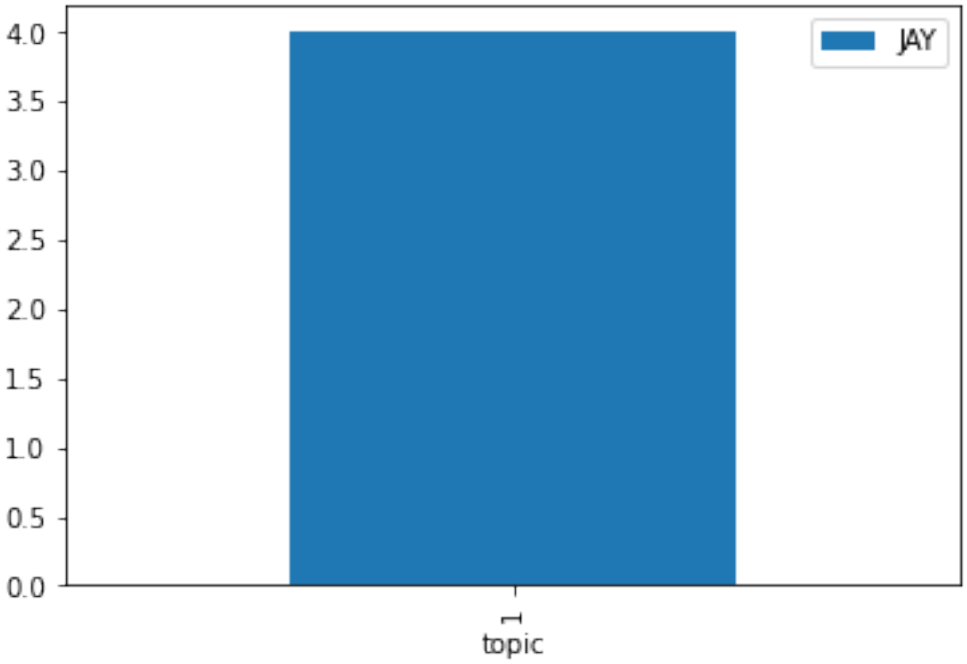
Using the model, for each essay assign it the most likely topic. For the undisputed papers, plot the histogram of this topic usage vs author.

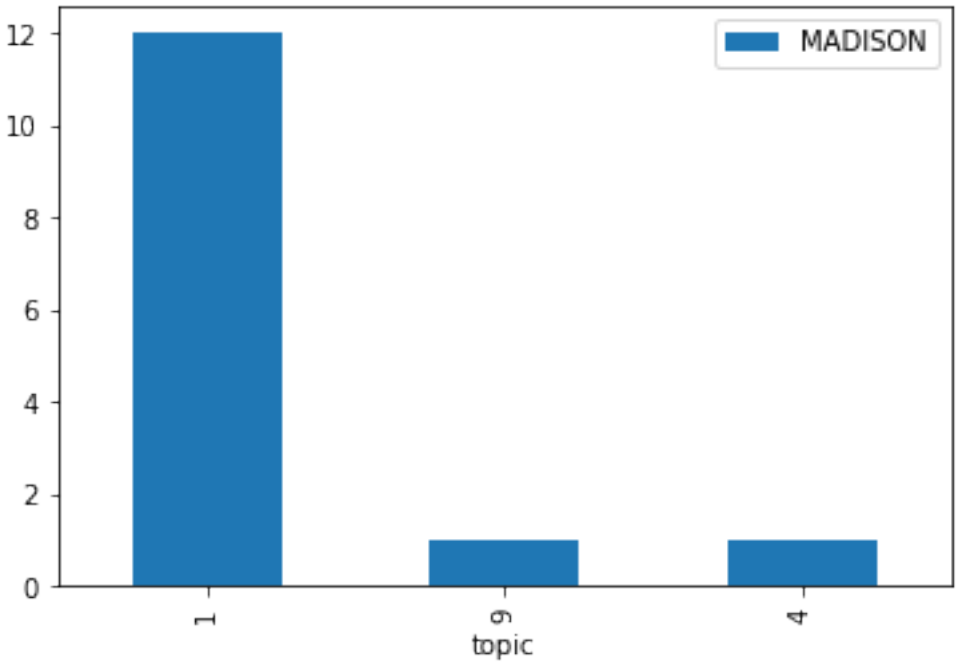
[416]:
```
#load authorship
authorship = pd.read_csv('authorship.csv', index_col = 0)
authorship.columns = ['author']
top_topics = np.argmax(res[1][1].mean().copy(), axis=1)
top_topics
authorship['topic'] = top_topics
counts = authorship.value_counts().to_frame('counts').reset_index()
```

[417]:
```
for label, grp in counts.groupby('author'):
    grp.plot.bar(x='topic', y='counts', label = label)
```




HAMILTON
25
20
15
10
5
0
1
6
9
8
topic


JAY
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
topic


MADISON
12
10
8
6
4
2
0
1
9
4
topic

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?

If you knew the author of each document in the training set, you could take the learned distribution over the topics for each author and compare new literature to these topic distributions. You could use a distance metric to grade the 'closeness' of the new document's topic proportions and then learned topic proportions from the training set.

1.3.2 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?

You could take a Probabilistic Graphical Model approach and add dependencies (edges) between word nodes. Then, you would solve for a set of parameters that are local to that word that capture small scale dependencies (i.e. markov blanket of the word). This would better account for dependencies and not make the exchangability assumption in LDA.

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.