MODULE 1: MAKING SENSE OF UNSTRUCTURED DATA

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 1.2 – LDA ANALYSIS

MIT | xPRO

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 1.2 – LDA ANALYSIS

Instructor: Tamara Broderick

TA: Qiuying (Giulia) Lai

In this document, we walk through some tips to help you with doing your own analysis on MIT EECS faculty data using stochastic variational inference on LDA. We provide some examples for the following programming environment: **Python**. You can find the full code for this project here: [4]. We cover the following:

1. Scraping your own dataset
2. Pre-processing the dataset
3. Implementing your own LDA code
4. Visualizing the results

1. Scraping your dataset

Using BeautifulSoup (https://www.crummy.com/software/BeautifulSoup/), and by analyzing the structure of the source code of arXiv, we could scrape the name list of MIT EECS faculty members. Using this information, we could list the query we send to arXiv. A possible format for the arXiv search for papers by authors is the following:

arxiv.org/find/(subject)/1/au:+(lastname)_(initial)/0/1/0/all/0/1

You could therefore adapt the names you scraped, and query through all the relevant arXiv search pages.

Within the arXiv source code, look for < class span=list-identifier >, which will give the identifier for the papers listed in your query results. Similarly look for the tag for the “Abstract” within each paper and scrape the abstract for each paper you find.

Note that you might want to scrape more information than you need and then do some local processing with the text you have instead.

2. Pre-processing the dataset

In the original work we have processed the data as raw documents as the dataset size was small. However if you want to use Matthew Hoffman’s original SVI code instead [3], that code takes a text file with a specific format. Once you have each abstract in a separate text file, you may find the following Python packages useful: **io, collections, nltk.** It is good practice to keep your dataset in its own folder, so **io** can be used to access that folder using a constant (relative) path. Read each file and use **nltk.tokenize** to tokenize each chunk of text. Use **collections** to process each abstract using a **Counter/Dictionary**, before writing the counts of words of each individual abstract as a line in the text file.

3. Implementing your own SVI-LDA code

Latent Dirichlet allocation (LDA) is a generative statistical model in natural language processing, and can be used to discover 'topics' in a large set of documents. This is first presented by David Blei, Andrew Ng, and Michael Jordan [1]. The key idea is that if we see a 'topic' as a collection of certain words, we can look at each document as a collection of topics, the proportion of each topic depends on the proportion of words in the document that are associated with that topic. For example, the 'sports' topic may consist of the words: tennis, football, gymnastics.

When given a set of documents, we can calculate the posterior distribution for the topics. In the original LDA paper, this is done using a coordinate descent algorithm for mean-field variational inference, and later on researchers also used Gibbs Sampling and expectation propagation.

In this tutorial we will be looking only at Stochastic Variational Inference for LDA. SVI was first published in 2013 by Matt Hoffman, David Blei, Chong Wang, and John Paisley [2]. Traditional coordinate-descent variational inference requires each update to be carried out with all of the data, and these updates become inefficient when the dataset gets large as each update scales linearly with the size of the data. The key idea with SVI is to update global variational parameters more frequently.

Using local and global parameters, and given the dataset with a known number of datapoints, we could randomly take 1 data point at a time, update the local parameter, and project the change into the global parameters. Like traditional coordinate-descent variational inference, this is done until the result converges, i.e., the change in the global parameters is smaller than a certain value.

The implementation we will be talking about is a naive implementation of the algorithm described in the original paper [2].

3.1 Variable Notation

Here we provide a brief overview of the input variables for LDA and SVI. Variables that can be set are the following:

- λ: what we want in the end (the posterior distribution for the topics for each word
- *vocab*: this is the overall vocabulary we will have in the docs
- K: this is the number of topics we want to get in the end
- D: this is the total number of documents
- α: parameter for per-document topic distribution
- η: parameter for per-topic vocab distribution

- τ: delay that down weights early iterations
- κ: forgetting rate, controls how quickly old information is forgotten; the larger the value, the slower it is.
- *max:iterations*: the number of maximum iterations the updates should go on for. We usually set a check such that if the difference in two consecutive values of λ is smaller than a certain value, we say the algorithm has converged. However, sometimes we could set this certain value too small, so we set a maximum iteration value to avoid updates running forever.

3.2 LDA Generative Model

We review the LDA generative model here. LDA assumes each document has *K* topics with different proportions. It models a corpus **w** of size *D* as follows:

- Draw distribution over vocabulary $\beta_k \sim \textit{Dirichlet}(\eta)$ for topics $k \in \{1 \ldots K\}$

- For each document $d \in \{1 \ldots D\}$:

  – Draw topic proportions $\theta_d \sim \textit{Dirichlet}(\alpha)$;

  – For each word $W_{d\,n}$ in the document:

    * Draw topic indicator $Z_{d\,n} \sim \textit{Multinomial}\ (\theta_d)$

    * Draw word $W_{d\,n} \sim \textit{Multinomial}\ (\beta_{Z_{d_n}})$

Note that this model follows the 'bag of words' assumption, such that given the topic proportions, each word drawn is independent of any other words in the document.

3.3 Code Walkthrough

Sometimes it is helpful to have your algorithm in a class, such that if you have multiple datasets, parameters and results, you keep them separated from each other. We can also initialize and here.

```
class SVILDA():
def __init__(self, vocab, K, D, alpha, eta, tau, kappa, docs, iterations):
self ._vocab = vocab
self ._V = len(vocab)
self ._K = K
self ._D = D
self ._alpha = alpha
self ._eta = eta
self ._tau = tau
self ._kappa = kappa
self ._lambda = 1* n.random.gamma(100., 1./100., (self._K, self._V))
self ._Elogbeta = dirichlet_expectation(self._lambda)
self ._expElogbeta = n.exp(self._Elogbeta)
self ._docs = docs
self.ct = 0
self._iterations = iterations
```

In this version we assume that the data has not been preprocessed, and also note that *numpy* was imported as *n*.

The function *dirichlet_expectation* is as follows, and it calculates the expectation for a beta distribution given its parameter, and is taken from Matthew Hoffman's original implementation on the Blei Lab GitHub space [3].

```
def dirichlet_expectation(alpha):
'''see onlineldavb.py by Matthew Hoffman'''
if (len(alpha.shape) == 1):
return (psi(alpha) - psi(n.sum(alpha)))
return (psi(alpha) - psi(n.sum(alpha, 1))[:, n.newaxis])
```

Now we can look at the local update and global update. The update function should closely follow the original algorithm, so we won't discuss in detail here as it is best to cross reference the original paper [2], but the key idea is to have the UpdateLocal function take in a parsed document, update the local variables, and then the UpdateGlobal function update the global parameters, like this:

```
def UpdateLocal(self, doc):
(words, counts) = doc
newdoc = []
N_d = sum(counts)
phi_d = n.zeros((self._K, N_d))
gamma_d = n.random.gamma(100., 1./100., (self._K))
Elogtheta_d = dirichlet_expectation(gamma_d)
expElogtheta_d = n.exp(Elogtheta_d)
for i, item in enumerate(counts):
    for j in range(item):
        newdoc.append(words[i])
assert len(newdoc) == N_d, "error"

for i in range(self._iterations):
    for m, word in enumerate(newdoc):
        phi_d[:, m] = n.multiply(expElogtheta_d, self._expElogbeta[:, word]) + 1e-100
        phi_d[:, m] = phi_d[:, m]/n.sum(phi_d[:, m])

    gamma_new = self._alpha + n.sum(phi_d, axis = 1)
    meanchange = n.mean(abs(gamma_d - gamma_new))
    if (meanchange < meanchangethresh):
        break

    gamma_d = gamma_new
    Elogtheta_d = dirichlet_expectation(gamma_d)
    expElogtheta_d = n.exp(Elogtheta_d)
    newdoc = n.asarray(newdoc)
    return phi_d, newdoc, gamma_d

    def UpdateGlobal(self, local_param, doc):
        lambda_d = n.zeros((self._K, self._V))
        for k in range(self._K):
            phi_dk = n.zeros(self._V)
            for m, word in enumerate(doc):
                phi_dk[word] += phi_d[k][m]
            lambda_d[k] = self._eta + self._D * phi_dk
    rho = (self.ct + self._tau) **(-self._kappa)
    self._lambda = (1-rho) * self._lambda + rho * lambda_d
    self._Elogbeta = dirichlet_expectation(self._lambda)
    self._expElogbeta = n.exp(self._Elogbeta)
```

And finally we could write an overall function to handle all updates taking into account the delay and forgetting rate, given the input variables:

```
def runSVI(self):

for i in range(self._iterations):

  randint = random.randint(0, self._D-1)

  print "ITERATION", i, " running document number ", randint doc =
  parseDocument(self._docs[randint],self._vocab) phi_doc, newdoc, gamma_d =
  self.updateLocal(doc) self.updateGlobal(phi_doc, newdoc)

  self.ct += 1
```

The integrated parseDocument function here is as

```
def parseDocument(doc, vocab): wordslist = list() countslist = list()
doc = doc.lower()

tokens = wordpunct_tokenize(doc)
for word in tokens: if word in vocab: wordtk = vocab[word]

if wordtk not in dictionary: dictionary[wordtk] = 1

else: dictionary[wordtk] += 1

wordslist.append(dictionary.keys())
countslist.append(dictionary.values())

return (wordslist[0], countslist[0])
```

There are also other functions in the code that are implemented and can be found on GitHub. For example, we could include code which allow us to ‘trace’ the change of values of certain variables in order to see the convergence of that value.

4 Visualizing the results

There are many ways to visualize the results once you have the per-topic vocab distributions. The first insight we could get is to look at what the most popular words are in each topic. To do this we just take our, and then by taking out each topic entry, we rank the words from highest probability to lower probabilities. This is how we arrive at the list of words for any topic in Figure 1. There are five such lists for the five topics.

Another simple visualization is to look at the topic distribution overall in the documents. To do this we could look at each document, infer how the topics are distributed for this document, and then sum up the probabilities for the topics across all the documents. This is how we arrived at the pie chart representing the topic proportions in Figure 1.

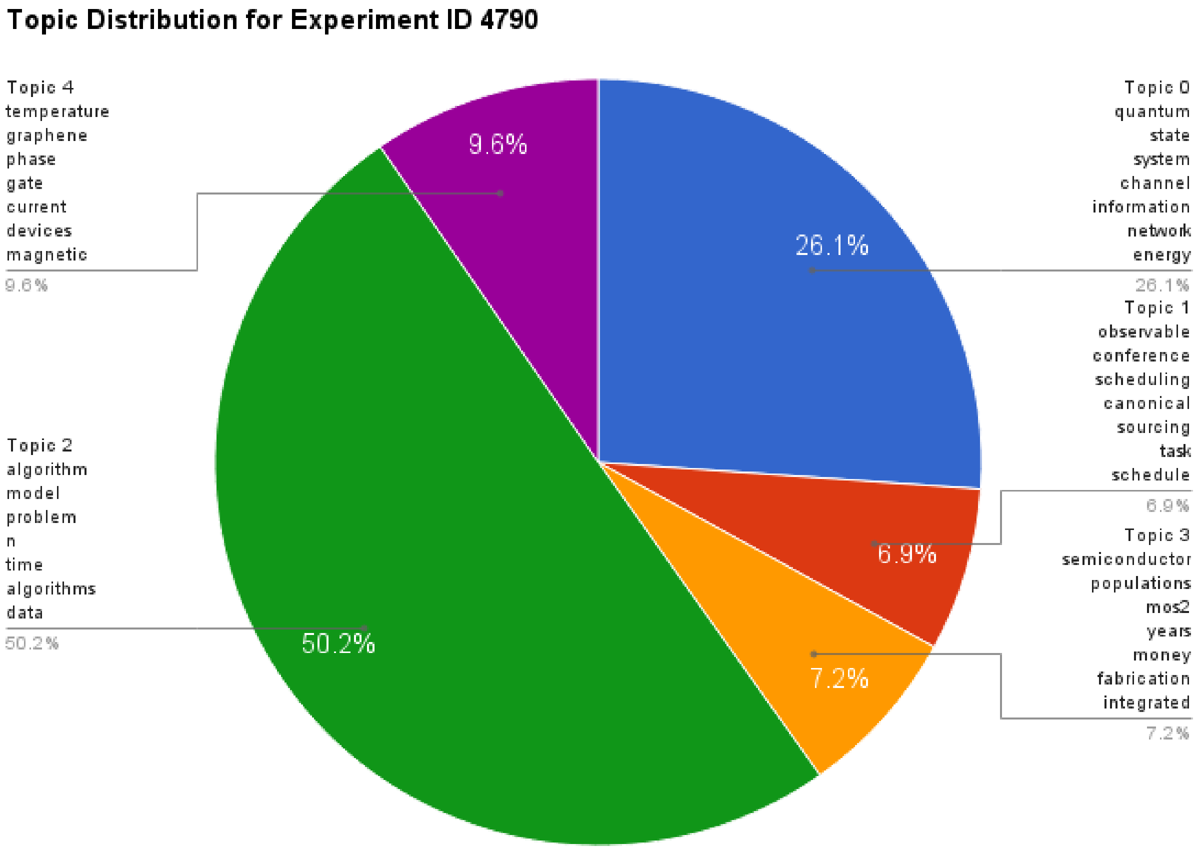

Figure 1: Topic distribution and most common words for one instance of *K* = 5.

Thirdly, we could also infer, using λ, the topic distribution of each document, and use this to guess the main focus of each publication scraped on arXiv. A way to do this is, for each topic, we sum up the normalized probability of each word (such that a less-used word would weigh less) in that topic over all words in the document, and then we compare this value across topics.

Finally, note that we have more than one paper per author and many authors per lab. To find the topic distribution within a lab, we can collect all the topic distributions across papers belonging to authors in that lab. This is how we arrive at the topic distributions for each of the four labs in Figure 2

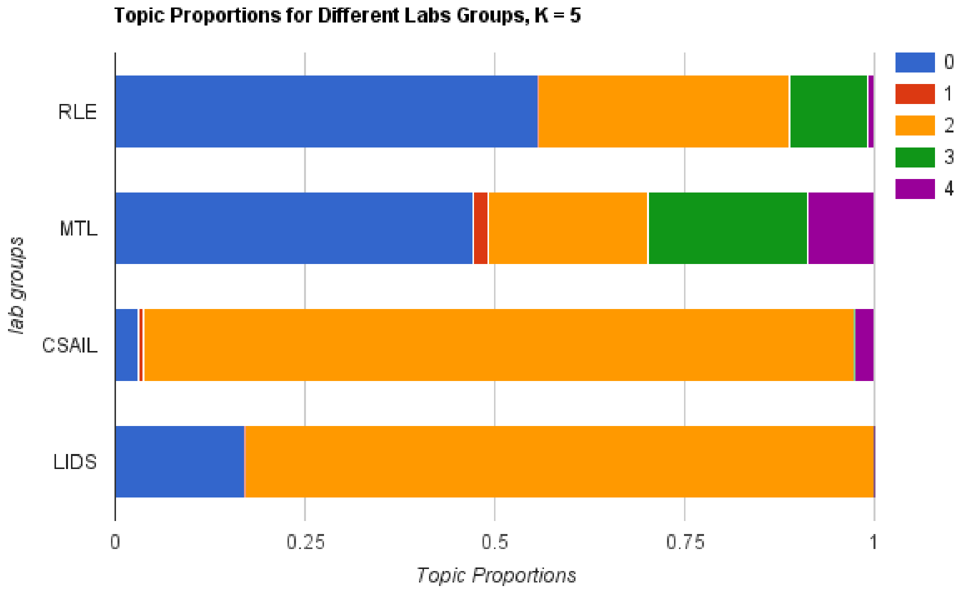

Figure 2: Lab Group interests for different values of *K*.

References

[1] Blei, D., Ng, A., and Jordan, M. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

[2] Hoffman, M., Blei, D., Wang, C., and Paisley, J. (2013). Stochastic variational inference.*Journal of Machine Learning Research*, 14:1303−1347.

[3] Hoffman, M. D. (2010). Online variational Bayes for latent Dirichlet allocation. https://github.com/blei-lab/onlineldavb

Lai, Q. (2016). Python implementation of Stochastic Variational Inference for LDA. https://github.com/qlai/stochasticLDA