CS5220: Assignment
Topic Modeling via Matrix Computation in Julia[*]

April 7, 2014

1 Introduction

Topic modeling is widely used to summarize major themes in large document collections. Topic modeling methods automatically extract topic clusters, represented by words and their relative intensities. In this assignment, we explore topic extraction using a type of non-negative matrix factorization of a matrix that captures correlations between words in a document collection. In this model, the matrix factors represent latent thematic structures.

Topic #	Top Words
1	**gas** station inside card case store location
2	thai **pad** curry spicy rice dish tea
3	pizza crust cheese sauce thin **pepperoni** toppings
4	**trail** parking lot park easy mountain phoenix view
5	**sum** wasn't minutes won't bad wanted phoenix
6	**yoga** classes class work favorite years hot
7	**donuts** coffee home day breakfast i'll shop

Table 1: First seven topic clusters from Yelp reviews

2 Model

Topic models are statistical models that represent documents in terms of shared topics. As these topics are not explicitly visible, the model assumes:

1. Each topic is a mixture of words and their relative intensities.[1]
2. Each document exhibits multiple topics with corresponding intensities.[2]

[*]Thanks to Moontae Lee and David Mimno for providing the first draft of this assignment and the data set. Any errors you find were probably subsequently introduced by DSB!

[1]In other words, a topic is a multinomial distribution over words.

[2]Similarly, a document is represented by a multinomial distribution over topics.

3. Each word in a document is generated with respect to its underlying topic.

Since only words in the documents are observable, our goal is to infer which words comprise each topic (i.e., topic-word distributions) and which topics comprise each document (i.e., document-topic distributions). The most popular approach to tackle this problem is called the Latent Dirichlet Allocation (LDA)[3]. We will use a less expensive approach based on *anchor words*[4].

Our model begins with the *word-word co-occurrence matrix* Q, which counts how likely each word jointly occurs with other words in the documents. Denote the number of distinctive words in the documents by V and the (predefined) number of topics by K. Our goal is to factorize this $V \times V$ matrix Q into a multiplication of several non-negative matrices so that we can parse hidden thematic structures from the resulting decomposition

$$Q \approx ARA^T, \tag{1}$$

where $A \in (\mathbb{R}^+)^{V\times K}$ is the *word-topic matrix* and $R \in (\mathbb{R}^+)^{K\times K}$ is the *topic-topic matrix.* An entry a_{ik} of the word-topic matrix can be thought of as probabilities of a word i given a topic k, or the *relative intensity* of word i to the topic k. An entry r_{ij} of the topic-topic matrix represents how much topic i is correlated with topic j. Our goal in this assignment is to recover A given Q.

3 Inference

Unfortunately, factoring Q as in (1) is not easy because we enforce the constraint that the columns of the word-topic matrix represent probability distributions (so are non-negative and sum to one). To simplify the computation, we make an assumption:

Assumption 1. *For each topic k, there is an **anchor word** i such that* $\begin{cases} A_{ik} > 0 \\ A_{ik'} = 0 \quad (\forall k' \neq k) \end{cases}$

Under this assumption, the inference of our model consists of the following steps:

1. Find the set of K anchor words given the matrix Q. (Say $S = \{s_1, \dots, s_K\}$)

2. Recover the word-topic matrix A with respect to the S.

3.1 Finding anchor words

Under the anchor-word assumption, every row vector of the row-normalized word-word co-occurrence matrix $\bar{Q}$ lies in the convex hull of the rows corresponding to anchor words. We find these representative rows with Algorithm 1[5]. The key step in this algorithm is the computation of the the (size of) the projection of all points onto the orthogonal complement of a current subspace span(S), which we can accomplish using either the Gram-Schmidt process or Householder transforms.

Algorithm 1 Find Anchor Words

```
In: the row-normalized word-word co-occurrence matrix $\bar{Q}$, the number of topics $K$
Out: the set of anchor words $S$
def Find-Anchors($\bar{Q}, K$)
  $S \leftarrow \{\bar{q}_i\}$ where $\bar{q}_i$ is the farthest point from the origin.
  for $k = 2$ to $K$ do
    $\bar{q}_i \leftarrow$ the point farthest from span($S$).
    $S \leftarrow S \cup \bar{q}_i$
  end for
  return $S$
```

Notation: $\bar{q}_i$ and $\bar{q}_j$ indicate some row vectors of the input matrix $\bar{Q}$

3.2 Recovering the word-topic matrix

Given the set of anchor words $S = \{s_1, ..., s_K\}$, now we need to figure out how to best approximate each non-anchor word vector as a convex combination of the anchor word vectors. Recall S was a greedy approximation, and the real data does not necessarily follow the anchor-word assumption. Thus the convex coefficients for each non-anchor word must be determined toward minimizing the gap between the original non-anchor word vector and the reconstructed convex combination. That is, for each word i,

$$\text{Find } (c_{i1}, \ldots, c_{iK}) \in [0,1]^K \text{ such that } \begin{cases} \text{minimizes } \|\bar{q}_i - \sum_{k \in S} c_{ik} \bar{q}_{s_k}\|_2^2 \\ \text{subject to } \sum_{k \in S} c_{ik} = 1 \text{ and } c_{ik} \geq 0. \end{cases} \tag{2}$$

Due to the constraints, this optimization is more complicated than an ordinary least squares fit; but it can be solved using Algorithm 2.

Note that c_{ik} computed above represents the conditional probability of the topic k given the word i. By using Bayes' theorem,

$$a_{ik} = p(word = i \mid topic = k) = \frac{p(topic = k \mid word = i) \cdot p(word = i)}{\sum_{i'} p(topic = k \mid word = i') \cdot p(word = i')} = \frac{c_{ik} w_i}{\sum_j c_{jk} w_j} \tag{3}$$

where

$$w_j = p(word = i) = \sum_j p(word_1 = i \mid word_2 = j) = \sum_j q_{ij}. \tag{4}$$

4 Dataset

We provide one data set in the repository, computed from the Yelp review corpus consisting of 20K documents and 1573 distinct words. This data set consists of a word co-occurrence matrix and a

[3]See this introduction: `https://www.cs.princeton.edu/~blei/papers/Blei2011.pdf`

[4]See `http://jmlr.org/proceedings/papers/v28/arora13.pdf` for details.

[5]For numerical linear algebra afficionados, this is equivalent to QR with column pivoting.

Algorithm 2 Constrained Minimization

```
In: The $V \times K$ matrix $T$, $V \times 1$ column-vector $b$
Out: the set of convex coefficients $x \in \Delta^K$ that minimizes $\|Tx - b\|^2$
def MINIMIZE-GAP($T, b$)
  $x \leftarrow (1/K, ..., 1/K)$ so that $\sum_{k=1}^{K} x_k = 1$
  $isConverge \leftarrow False$
  while not $isConverge$ do
    $p \leftarrow \nabla\|Tx - b\|_2^2 = 2T^T(Tx - b) \in \mathbb{R}^K$        ▷ compute the gradient
    for $k = 1$ to $K$ do
      $x_k \leftarrow x_k e^{-\eta_t p_k}$        ▷ perform a component-wise multiplicative update
    end for
    $x \leftarrow x/|x|_1$        ▷ projection onto the simplex
    $p' \leftarrow \nabla\|Tx - b\|_2^2$        ▷ recompute the gradient
    if $(p' - p'_{min}\mathbf{1})^T x < \epsilon$ then
      $isConverge \leftarrow True$        ▷ test a convergence
    end if
  end while
  return $S$
```

Notation: Δ^K indicates the simplex and p'_{min} indicates the minimum component of p'.

list of words. You can run the inference on the Yelp data set by running the script `driver.jl`; the anchor words and the top twenty words for each topic will be written file `topics.txt`.
You may also want to try some other data sets from the UCI machine learning data set repository that involve larger vocabularies. To download the relevant data, run

```
make fetch_kos
make fetch_nips
make fetch_enron
```

You can compute a corresponding topics word list by running (for example)

```
include("anchor_words.jl")
driver_uci("kos")
```

at the Julia prompt.

5 Your assignment

1. You are given a reference implementation that solves the constrained minimization problem using an accurate but expensive algorithm. Make sure you can run this implementation on the compute nodes. On C4, try this:

```
cd cs5220/topics
module load julia
csub julia driver.jl
```

Take note of the time required for the different phases of the computation. You may also want to try a somewhat larger data set.

2. Implement the exponentiated gradient algorithm (Algorithm 2) as an alternative to the current simplex strategy. You may want to bound the number of iterations when you see slow convergence. This involves filling in the `simplex_nnls_eg` routine in the code. How much does this improve the speed? If you are feeling ambitious and know something about this subject, you may also choose to implement an alternative algorithm.

3. Parallelize the algorithm. I recommend focusing on the most expensive computation in compute_A. This is embarrassingly parallel, but you may find that you need to think about data locality. For the purposes of this assignment, I recommend submitting your Julia script with `ompsub` in order to reserve the number of processors you need; but if you decide you want more than the eight processors available on one node, you can use start Julia on the front node and run the `addprocs_htc` command to launch jobs on the remote nodes.

4. Analyze the performance of the parallel algorithm. Where are the bottlenecks? What sort of parallel speedup do you see? How fast can you get the algorithm to go? You may also want to comment on the tradeoffs involved in a sloppy solve.

Your final submission should include:

- A modified version of the `anchor_topics.jl` code.
- The `topics.txt` file produced by your final implementation
- A writeup. Your writeup must include a speedup plot (speedup vs number of processors) with at least seven processors and a description of bottlenecks (observed via the profiler or tic/toc). Make sure you also report the time taken for phases of the code that you have not parallelized (including I/O).