Parallel PLSI on Spark

Hongwei Wang, Bicheng Zhang

Department of Computer Science
University of Illinois at Urbana-Champaign
{hwang172, zhang368}@illinois.edu

1 Introduction

Probabilistic Latent Semantic Indexing (PLSI) [5] is a category of topic models, which aims to find hidden topics in a corpus of documents. It is widely used in information retrieval, natural language processing and machine learning. As the number of documents increase, it is necessary to parallelize the computation in multiple nodes, and a popular way of doing it is MapReduce [4], a distributed framework proposed by Google and open sourced by Apache Hadoop [1].

However, the MapReduce implementation in Hadoop is not that efficient for PLSI computation. The reason is that PLSI is an iterative process, and the output of a Reduce phase will become the input of next Map phase, while in Hadoop, it always writes the output of Map and Reduce to the disk, which incurs a large overhead between each iteration.

Recently, a new distributed framework called Spark [2], was proposed to solve the problem of iterative MapReduce jobs by introducing a new distributed memory abstraction called resilient distributed datasets (RDD) [6], which supports in-memory computation and efficiently data reuse. It maintains a fault tolerant memory cache on each node so that the output of MapReduce can keep in memory for next iteration.

Since currently Spark does not support PLSI [3]. The goal of our project is to implement PLSI on Spark.

The rest of the report is organized as follows. Section 2 introduces PLSI and its computation. Section 3 shows the MapReduce algorithm of implementing PLSI. Section 4 introduces the experiments and section 5 shows the conclusion and future work.

2 PLSI

PLSI models the the probability of each observed document and word pairs \((d, w)\) as a mixture of conditional independent multinomial distribution with each topic as a hidden variable. PLSI assumes the process of generating a pair of \((d, w)\) as follows:

- Choose a document with the probability of \(P(d)\).
- Given the document \(d\), choose a topic \(z\) with probability \(P(z|d)\).
- Given the topic \(z\), choose a word \(w\) with probability \(P(w|z)\).
So the joint probability of a \((d, w)\) pair is

\[
P(d, w) = P(d)P(w|d) = P(d) \sum_z P(w|z)P(z|d)
\]

The joint distribution for the whole corpus is

\[
\prod_{d,w} P(d, w) = \prod_{d,w} P(d)P(w|d) = \prod_{d,w} P(d) \sum_z P(w|z)P(z|d)
\]

We are interested in estimating the parameters \(P(z|d)\) and \(P(w|z)\) given the corpus. \(P(z|d)\) represents the topic distribution in a document, and \(P(w|z)\) represents the word distribution within a topic.

Let \(\theta\) denotes all the parameters, and the log maximum likelihood is

\[
\log L(\theta) = \sum_{d,w} n(d, w)\log P(d, w; \theta)
\]

\[
= \sum_{d,w} n(d, w)\log P(w|d; \theta)P(d)
\]

\[
= \sum_{d,w} n(d, w)\log P(w|d; \theta) + \sum_{d,w} n(d, w)\log P(d)
\]

\[
= \sum_{d,w} n(d, w)\log \sum_z P(w|z)P(z|d) + \sum_{d,w} n(d, w)\log P(d)
\]

Since

\[
\sum_{d,w} n(d, w)\log P(d)
\]

does not contain \(\theta\), we only need to find

\[
\theta_{ML} = \arg \max_\theta \sum_{d,w} n(d, w)\log \sum_z P(w|z)P(z|d)
\]
Here we use expectation-maximization (EM) algorithm. The iterative procedure is defined as \( \theta_t = \arg \max_{\theta} Q(\theta, \theta_{t-1}) \), where

\[
Q(\theta, \theta_{t-1}) = \sum_{d, w} n(d, w) \sum_{z} P(z|d, w; \theta_{t-1}) \log P(w|d; \theta)
\]

\[
= \sum_{d, w} n(d, w) \sum_{z} P(z|d, w; \theta_{t-1}) [\log P(w|z) + \log P(z|d)]
\]

**E step:**

\[
P(z|d, w; \theta_{t-1}) = \frac{P_{t-1}(z|d) P_{t-1}(w|z)}{\sum_{z'} P_{t-1}(z'|d) P_{t-1}(w|z')}
\]

We can compute \( Q(\theta, \theta_{t-1}) \) with the above \( P(z|d, w; \theta_{t-1}) \).

**M step:**

We need to compute

\[
\theta_t = \arg \max_{\theta} Q(\theta, \theta_{t-1})
\]

where \( \theta_t \) denotes the parameters \( P(w|z) \) and \( P(z|d) \) for each \( w, z, d \) with the constraint of

\[
\sum_w P(w|z) = 1 \quad (1)
\]

\[
\sum_z P(z|d) = 1 \quad (2)
\]

To solve the optimization problem, we construct the Lagrange function by introducing Lagrange multipliers.

\[
H = Q(\theta, \theta_{t-1}) + \sum_z \alpha_z (1 - \sum_w P(w|z)) + \sum_d \beta_d (1 - \sum_z P(z|d))
\]

We have

\[
\frac{\partial H}{\partial P(w|z)} = \sum_d n(d, w) P(z|d, w; \theta_{t-1}) - \alpha_z
\]

\[
\frac{\partial H}{\partial P(z|d)} = \sum_w n(d, w) P(z|d, w; \theta_{t-1}) - \beta_d
\]

Set

\[
\frac{\partial H}{\partial P(w|z)} = 0
\]

\[
\frac{\partial H}{\partial P(z|d)} = 0
\]

i.e.

\[
\sum_d n(d, w) P(z|d, w; \theta_{t-1}) = \frac{P(w|z)}{\alpha_z} = 0
\]
$$\sum_w n(d, w) P(z|d, w; \theta_{t-1}) - \beta_d = 0$$

(4)

With (1)(2)(3)(4), we can compute

$$P(w|z) = \frac{\sum_d n(d, w) P(z|d, w; \theta_{t-1})}{\sum_{d, w'} n(d, w') P(z|d, w'; \theta_{t-1})}$$

$$P(z|d) = \frac{\sum_w n(d, w) P(z|d, w; \theta_{t-1})}{\sum_{w', z} n(d, w) P(z'|d, w; \theta_{t-1})} = \frac{\sum_w n(d, w) P(z|d, w; \theta_{t-1})}{\sum_w n(d, w)}$$

### 3 MapReduce Algorithm for PLSI

From section 2, we know that the EM algorithm to estimate the parameters of PLSI is an iterative process. We can easily write the driver program of MapReduce in algorithm 1, which is similar to local PLSI computation.

#### Algorithm 1 Parallel PLSI

```
initialize $P(z|d), P(w|z)$
for each iteration do
  carry out map reduce jobs
  update $P(w|z), P(d|z)$
  check likelihood
end for
output $P(w|z), P(d|z)$
```

The key issue is how to parallelize and distribute the computation in MapReduce jobs. Based on our observation, the computation within each document is independent given $P(z|d)$ and $P(w|z)$.

In the Map phase, we can compute $n(d, w) * P(z|d, w; \theta_{t-1})$, $P(d|z)$ and partial likelihood based on previous round of parameters $P(z|d)$ and $P(w|z)$.

In the Reduce phase, we can sum the the output of Map phase to get the total $P(z|d, w; \theta_{t-1})$, $P(z|d)$ and the global likelihood. Then the driver program will update the parameters of $P(z|d)$ and $P(w|z)$ and start next iteration until the likelihood converges.

Algorithm 2 and 3 show the Map phase and Reduce phase respectively.

### 4 Experiment

In our experiment, we set up 5 instances in Amazon EC2. Each instance is of type m3.large, with 2 vCPUs, 7.5 GB memory and 32 GB SSD. The latest Spark (version 0.9.1) is set up on all instances. One is the master node and the other four are worker nodes. We also install Hadoop 2.2.0 since Spark supports reading files from HDFS. The corpus we use is a sample of Wikipedia data, with 10,000 documents and 58750 unique words after removing stop words.
Algorithm 2 Map(key, value)

**Input:** key = docIndex, value = \( n(d, w), P(z|d), P(w|z) \)

**Output1:** key = topicIndex-w-wordIndex, value = \( n(d, w) \times P(z|d, w; \theta_{t-1}) \)

**Output2:** key = topicIndex-d-docIndex, value = \( P(d|z) \)

**Output3:** key = 1, value = likelihood

initialize an array \( \text{sum\_words} \) to store \( \sum_w n(d, w) \times P(z|d, w; \theta_{t-1}) \) for each topic

\[ \text{likelihood} = 0 \]

for Each word in vocabulary do
  // E-step
  for each topic do
    Compute \( P(z|d, w; \theta_{t-1}) = P(z|d) \times P(w|z) \)
  end for
  // M-step
  for each topic do
    normalize \( P(z|d, w; \theta_{t-1}) \)
    compute \( n(d, w) \times P(z|d, w; \theta_{t-1}) \)
    emit output1
    \( \text{sum\_words}[z] = \text{sum\_words}[z] + n(d, w) \times P(z|d, w; \theta_{t-1}) \)
    \( \text{likelihood} = \text{likelihood} + n(d, w) \times P(z|d, w; \theta_{t-1}) \times [\log P(z|d) + \log P(w|z)] \)
  end for
for each topic do
  compute \( P(z|d) = \text{sum\_words}[z] / \sum_w n(d, w) \)
  emit output2
end for
emit output3

Algorithm 3 Reduce(key, values)

**Input:** (key, values)

**Output:** (key, sum)

\[ \text{sum} = 0 \]

for each value in values do
  \( \text{sum} += \text{value} \)
end for
emit (key, sum)
We implement the Parallel PLSI algorithm with Python.

In our experiment, we select 50 topics to run our PLSI algorithm on the corpus, and here are some partial results. We have chosen the top 10 frequent words within the following 5 topics.

<table>
<thead>
<tr>
<th>Space</th>
<th>Politics</th>
<th>Geophysics</th>
<th>Art</th>
<th>Australia</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>countries</td>
<td>water</td>
<td>music</td>
<td>Australia</td>
</tr>
<tr>
<td>stars</td>
<td>united</td>
<td>earth</td>
<td>people</td>
<td>south</td>
</tr>
<tr>
<td>light</td>
<td>country</td>
<td>mars</td>
<td>plays</td>
<td>land</td>
</tr>
<tr>
<td>pole</td>
<td>city</td>
<td>air</td>
<td>wrote</td>
<td>waves</td>
</tr>
<tr>
<td>universe</td>
<td>war</td>
<td>oxygen</td>
<td>written</td>
<td>food</td>
</tr>
<tr>
<td>red</td>
<td>world</td>
<td>planet</td>
<td>pop</td>
<td>large</td>
</tr>
<tr>
<td>north</td>
<td>union</td>
<td>things</td>
<td>composers</td>
<td>found</td>
</tr>
<tr>
<td>earth</td>
<td>queue</td>
<td>scientists</td>
<td>art</td>
<td>continent</td>
</tr>
<tr>
<td>energy</td>
<td>south</td>
<td>life</td>
<td>instrument</td>
<td>ocean</td>
</tr>
<tr>
<td>star</td>
<td>European</td>
<td>atmosphere</td>
<td>theater</td>
<td>bird</td>
</tr>
</tbody>
</table>

Table 1. Top 10 frequent words within each topic

5 Conclusion and Future Work

In this report, we introduce PLSI and its MapReduce implementation on Spark. For the future work, there is still large room for performance optimization based on the work flow (the MapReduce algorithm) and Spark specific features. It can also be implemented on Hadoop as a comparison with Spark.

References