Implementation Explained: Partition Optimization in sequential Information Bottleneck

1 Abstract

This document describes the enclosed implementation of the optimization step in the text clustering algorithm *sequential Information Bottleneck*. We present a mathematical development of the formulas from the original paper, which allows us to reach an efficient and simple computation of the optimization step, with focus on the case of sparse vector representation. Based on this development, the enclosed open-source implementation of the algorithm is more practical for real-world use-cases.

2 Introduction

The sequential Information Bottleneck (sIB) [3] is a text clustering algorithm that shows strong results on standard benchmark datasets, such as [1], compared to the more commonly used Lloyd's K-Means [2]. However, sIB has never been as popular as K-Means, partly because in the trade-off between quality and speed, sIB has a very strong tendency towards better quality, at the expense of longer run-time, while K-Means is a more balanced choice. This makes sIB unsuitable in many circumstances, especially when speed is a high priority.

The goal of this work is to present a mathematical development of the formulas presented in sIB's original paper, and to explain how this development is used for a new efficient implementation of sIB. This is achieved without scarifying the quality of the clustering analysis that the algorithm generates. The new implementation makes sIB more practical for real-world applications and is available as an open-source in Python and C++.[1]

3 sIB in a Nutshell

In contrast to algorithms such as K-Means, where the objective function and the distance function that they employ are two separate entities, in the case of sIB, the objective of in-cluster information maximization dictates the formula of the distance function by means of a mathematical derivation. Thus, sIB benefits from using an optimal distance function for the objective it uses, which turns out to be a weighted Jensen-Shannon divergence. However, this distance function is also more compute-intensive compared to distance functions such as Euclidean or Cosine distance.

Moreover, sIB is a sequential algorithm, which means that (a) before computing the new cluster for a sample, sIB withdraws the sample from its current cluster to prevent that sample from biasing the distance function towards keeping it in the same cluster, and (b) sIB updates the centroids while iterating over the samples and not only at the end of the iteration.

Overall, while iterating over the samples, every sample is withdrawn from its cluster, the centroid of that cluster is updated, a new cluster is selected for the sample using the weighted JS divergence distance function, then the sample is added to the new cluster and eventually the centroid of the new cluster is updated. In total, sIB performs $2 \cdot n\_samples$ centroid updates during a full iteration, while Lloyd's K-Means performs only $n\_clusters$ updates.

[1] `https://github.com/IBM/sib`

By taking into account both the higher computational intensity of sIB's distance function and the significantly higher number of centroid updates that it performs, it is easy to see why sIB is inherently slower than algorithms such as Lloyd K-Means. However, as we show here, an efficient implementation is employable and closes the run-time gap between sIB and K-Means to a large extent.

4 Bag-of-Words and Vector Representation

sIB adopts the *Bag-of-Words* (BoW) model, in which texts are represented using vectors over a vocabulary of terms, such as unigrams. The vocabulary is a corpus-level concept – an ordered list that aims to include the terms that are assumed to be meaningful for comparing texts in this corpus. When a text is represented in the BoW model, a vector is used to specify the frequency of each vocabulary item in that text. In our implementation, we assume that the vectors of texts are raw (un-normalized) count vectors.

Typically, the number of unique terms found in a specific text is smaller than the vocabulary size by a large margin. Therefore, it is more efficient to represent texts using sparse vector representations, both in terms of memory usage and computation workload. In the sparse representation, it is sufficient to hold the list of ids of vocabulary items found in the text and their frequency, rather than an array of the size of the full vocabulary in which most of the values are zero. However, there are circumstances in which it is preferable to use a dense representation for all vectors. Thus, the two representations should be supported.

sIB is a centroid-based clustering algorithm, and as such it represents centroids using vectors. The centroid of every cluster is constructed from the set of vectors that are associated with that cluster, and therefore centroid vectors refer to a large part of the vocabulary. As a result, such vectors are typically represented as dense vectors, regardless of the format of the vectors of individual texts.

5 Pseudo-Code and Focus

The pseudo-code of the algorithm main-loop is given in Figure 1 (quoted from [3]) and outlines the sequential workflow in which sIB works. In this code, X is the vectors of the samples to cluster, K is the number of clusters, n is the number of (random) initializations, $maxL$ is the maximal number of iterations per initialization, and ϵ is a lower bound threshold on the cluster updates for continuing to another iteration.[2] In addition, t is used as a cluster identifier, and x as a sample identifier.

In this document we will focus on the inner *for*-loop of the pseudo-code, which is the partition optimization part. This part contains the three statements repeated below:

1. Draw x_j out of $t(x_j)$
2. Compute $t^{new}(x_j) = \arg\min_{t'} d_(\{x_j\}, t')$
3. Merge x_j into $t^{new}(x_j)$

Statement 2 is at the core of the partition optimization and is the most intense to compute. Therefore, we start by investigating it in Section 6, and after that we move on to statements 1 and 3 in Section 7.

6 Computing $t^{new}(x)$

Let us rephrase the equation for computing $t^{new}(x)$ in a simpler form in equation (1).

$$t^{new}(x) = \arg\min_{t} d_F(x, t) \tag{1}$$

According to (1), we assign a sample x to the cluster t that minimizes the function d_F, given in (2).

$$d_F(x, t) = (p(x) + p(t)) \cdot JS(p(y|x), p(y|t)) \tag{2}$$

[2] Using several random initializations is a common practice with many clustering algorithms, as each initialization converges only to a local maximum/minimum.

Input:

$|X|$ objects to be clustered
Parameters: $K,\ n,\ maxL,\ \epsilon$

Output:

A partition T of X into K clusters

Main Loop:

```
For i = 1, ..., n
    T_i ← random partition of X.
    c ← 0, C ← 0, done = FALSE
    While not done
        For j = 1, ..., |X|
            draw x_j out of t(x_j)
            t^new(x_j) = arg min_{t'} d_(){x_j}, t')
            If t^new(x_j) ≠ t(x_j) then c ← c + 1
            Merge x_j into t^new(x_j)
        C ← C + 1
        if C ≥ maxL or c ≤ ε · |X| then
            done ← TRUE
T ← arg max_{T_i}(T_i)
```

Figure 1: Pseudo-code for the sequential clustering algorithm.

where JS is a weighted *Jensen-Shannon divergence* defined with weights $pi1$ and $pi2$ as follows:

$$pi1_{x,t} = \frac{p(x)}{p(x)+p(t)} \tag{3}$$

$$pi2_{x,t} = \frac{p(t)}{p(x)+p(t)} \tag{4}$$

$$JS(p(y|x), p(y|t)): \tag{5}$$

$$average = pi1 \cdot p(y|x) + pi2 \cdot p(y|t)) \tag{6}$$

$$kl1 = KL(p(y|x), average) \tag{7}$$

$$kl2 = KL(p(y|t)), average) \tag{8}$$

$$\text{return } pi1 \cdot kl1 + pi2 \cdot kl2 \tag{9}$$

and KL is the *Kullback-Leibler divergence* defined as:

$$KL(u,v) = \sum_i u[i] \cdot log(\frac{u[i]}{v[i]}) \tag{10}$$

Intuitively, when assigning the sample x to a new cluster, t^{new}, we look at the distribution of the vocabulary over x ($p(y|x)$) and compare it to the distribution of the vocabulary over each cluster's centroid ($p(y|t)$) using the weighted JS divergence. The cluster t^{new} is the cluster in which the distribution of the vocabulary in its centroid is the closest to the distribution of the vocabulary in x.

Definitions

Let XY be the joint count matrix of samples, where sample x_i is represented by the vector at raw i. We indicate the vector of sample x by the notation $XY[x]$.[3] In addition, let t be a cluster and let $x_t^0, ..., x_t^n$ be the samples associated with it. We define the set of notations in Table 1.

[3]In the case of spare vector representation, XY is a sparse matrix.

Notation	Definition	Description	Type
XY_{sum}	$\sum_{x,y} XY[x][y]$	The sum of values in the count matrix	Scalar
x_{sum}	$\sum_{y} XY[x][y]$	The sum of values in the vector of x	Scalar
$p(x)$	$\frac{x_{sum}}{XY_{sum}}$	The probability of x	Scalar
$p(y\|x)$	$\frac{XY[x]}{x_{sum}}$	The probability vector representing x	Vector
t_{sum}	$\sum_{i}^{n} x^{i}_{tsum}$	The sum of values in all vectors of samples associated with t	Scalar
$p(t)$	$\frac{t_{sum}}{XY_{sum}}$	The probability of t	Scalar
$t_{centroid}$	$\sum_{i}^{n} XY[x^{i}_{t}]$	The sum of vectors of samples associated with t	Vector
$p(y\|t)$	$\frac{t_{centroid}}{t_{sum}}$	The probability vector representing the centroid of t	Vector

Table 1: Basic Definitions

From the above definitions we can derive a set of equations:

$$p(x) + p(t) = \frac{x_{sum}}{XY_{sum}} + \frac{t_{sum}}{XY_{sum}} = \frac{x_{sum} + t_{sum}}{XY_{sum}} \tag{11}$$

$$\frac{1}{pi2} = \frac{p(x) + p(t)}{pt} = (p(x) + p(t)) \cdot \frac{1}{pt} = \frac{x_{sum} + t_{sum}}{XY_{sum}} \cdot \frac{XY_{sum}}{t_{sum}} = \frac{x_{sum} + t_{sum}}{t_{sum}} \tag{12}$$

$$p(y|t) \cdot pi2 = \frac{t_{centroid}}{t_{sum}} \cdot \frac{p(t)}{p(x) + p(t)} = \frac{t_{centroid}}{t_{sum}} \cdot \frac{t_{sum}}{XY_{sum}} \cdot \frac{XY_{sum}}{x_{sum} + t_{sum}} = \frac{t_{centroid}}{x_{sum} + t_{sum}} \tag{13}$$

$$p(x) \cdot p(y|x) + p(t) \cdot p(y|t) = \frac{x_{sum}}{XY_{sum}} \cdot \frac{XY[x]}{x_{sum}} + \frac{t_{sum}}{XY_{sum}} \cdot \frac{t_{centroid}}{t_{sum}} = \frac{XY[x] + t_{centroid}}{XY_{sum}} \tag{14}$$

$$\begin{aligned} pi1 \cdot p(y|x) + pi2 \cdot p(y|t) = \frac{p(x)}{p(x) + p(t)} \cdot p(y|x) + \frac{p(t)}{p(x) + p(t)} \cdot p(y|t) &= \frac{p(x) \cdot p(y|x) + p(t) \cdot (y|t)}{p(x) + p(t)} \\ &= \frac{XY[x] + t_{centroid}}{x_{sum} + t_{sum}} \end{aligned} \tag{15}$$

In the computations below we use s and c as abbreviations of $p(y|x)$ and $p(y|t)$ respectively. Since $p(y|x)$ and $p(y|t)$ are probability vectors, we get:

$$\sum_{i} s[i] = \sum_{i} p(y|x)[i] = 1 \tag{16}$$

$$\sum_{i} c[i] = \sum_{i} p(y|t)[i] = 1 \tag{17}$$

In addition, we will use the following equations:

$$\arg\min_{t} f(x) + g(x,t) = \arg\min_{t} g(x,t) \tag{18}$$

$$\arg\min_{t} f(x) \cdot g(x,t) = \arg\min_{t} g(x,t) \tag{19}$$

6.1 Computation for Sparse Vector Representation

Let us assume that x is represented by a sparse vector in which x_{ind} is the list of indices of vocabulary items that are found in x and $x[i] = XY[x][i]$ is the frequency of the item at index i. It follows that:

$$\forall i \in x_{ind}.x[i] > 0 \tag{20}$$

$$\forall i \notin x_{ind}.x[i] = 0 \tag{21}$$

It follows that:

$$\forall i \in x_{ind}.p(y|x)[i] = \frac{XY[x][i]}{x_{sum}} = \frac{x[i]}{x_{sum}} > 0 \tag{22}$$

$$\forall i \notin x_{ind}.p(y|x)[i] = \frac{XY[x][i]}{x_{sum}} = \frac{x[i]}{x_{sum}} = 0 \tag{23}$$

Computation of $kl1$

$$kl1 = KL(s, pi1 \cdot s + pi2 \cdot c) \tag{24}$$

$$= \sum_i s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \tag{25}$$

$$= \Big[\sum_{i \in x_{ind}} s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + \Big[\sum_{i \notin x_{ind}} s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] \tag{26}$$

$$= \sum_{i \in x_{ind}} s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \quad \text{by (23)} \tag{27}$$

Computation of $kl2$

$$kl2 = KL(c, pi1 \cdot s + pi2 \cdot c) \tag{28}$$

$$= \sum_{i=0}^{d} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \tag{29}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + \Big[\sum_{i \notin x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] \tag{30}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + \sum_{i \notin x_{ind}} c[i] \cdot log(\frac{c[i]}{pi2 \cdot c[i]}) \quad \text{by (23)} \tag{31}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + \sum_{i \notin x_{ind}} c[i] \cdot log(\frac{1}{pi2}) \tag{32}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + log(\frac{1}{pi2}) \cdot \sum_{i \notin x_{ind}} c[i] \tag{33}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + log(\frac{1}{pi2}) \cdot (\sum_i c[i] - \sum_{i \in x_{ind}} c[i]) \tag{34}$$

$$= \Big[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})\Big] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \quad \text{by (17)} \tag{35}$$

Computation of d_F

$$
\begin{aligned}
d_F(x,t) &= (p(x)+p(t)) \cdot JS(p(y|x), p(y|t)) && (36)\\
&= (p(x)+p(t)) \cdot JS(s,c) && (37)\\
&= (p(x)+p(t))(pi1 \cdot kl1 + pi2 \cdot kl2) && (38)\\
&= (p(x)+p(t))(\frac{p(x)}{p(x)+p(t)} \cdot kl1 + \frac{p(t)}{p(x)+p(t)} \cdot kl2) && (39)\\
&= p(x) \cdot kl1 + p(t) \cdot kl2 && (40)\\
&= p(x) \cdot \Big[\sum_{i \in x_{ind}} s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (41)\\
&\quad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \Big] && (42)
\end{aligned}
$$

Computation of $t^{new}(x)$

$$
\begin{aligned}
t^{new}(x) &= \arg\min_t d_F(x,t)\\
&= \arg\min_t \Big(p(x) \cdot \Big[\sum_{i \in x_{ind}} s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (43)\\
&\qquad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \Big] \Big)\\
&= \arg\min_t \Big(\Big[p(x) \cdot \sum_{i \in x_{ind}} s[i] \cdot log(s[i]) \Big] + \Big[p(x) \cdot \sum_{i \in x_{ind}} s[i] \cdot log(\frac{1}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (44)\\
&\qquad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \Big] \Big)
\end{aligned}
$$

Based on equation (18) :

$$
\begin{aligned}
&= \arg\min_t \Big(p(x) \cdot \Big[\sum_{i \in x_{ind}} s[i] \cdot log(\frac{1}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (45)\\
&\qquad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])})] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \Big] \Big)\\
&= \arg\min_t \Big(\Big[\sum_{i \in x_{ind}} (p(x) \cdot s[i] + p(t) \cdot c[i]) \cdot log(\frac{1}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (46)\\
&\qquad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(c[i])] + log(\frac{1}{pi2}) \cdot (1 - \sum_{i \in x_{ind}} c[i]) \Big] \Big)\\
&= \arg\min_t \Big(\Big[\sum_{i \in x_{ind}} (p(x) \cdot s[i] + p(t) \cdot c[i]) \cdot log(\frac{1}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \Big] + && (47)\\
&\qquad p(t) \cdot \Big[[\sum_{i \in x_{ind}} c[i] \cdot log(c[i])] - [\sum_{i \in x_{ind}} c[i] \cdot log(\frac{1}{pi2})] + log(\frac{1}{pi2}) \Big] \Big)
\end{aligned}
$$

$$= \arg\min_t \left(\left[\sum_{i \in x_{ind}} (p(x) \cdot s[i] + p(t) \cdot c[i]) \cdot log(\frac{1}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] + p(t) \cdot \left[\left[\sum_{i \in x_{ind}} c[i] \cdot log(c[i] \cdot pi2) \right] + log(\frac{1}{pi2}) \right] \right) \tag{48}$$

$$= \arg\min_t \left(\left[\sum_{i \in x_{ind}} (p(x) \cdot p(y|x) + p(t) \cdot p(y|t))[i] \cdot log(\frac{1}{(pi1 \cdot p(y|x) + pi2 \cdot p(y|t))[i]}) \right] + p(t) \cdot \left[\left[\sum_{i \in x_{ind}} p(y|t)[i] \cdot log((p(y|t) \cdot pi2)[i]) \right] + log(\frac{1}{pi2}) \right] \right) \tag{49}$$

Based on equations (12), (13), (14) and (15), and the definition of $p(t)$ and $p(y|t)$ from Table 1:

$$= \arg\min_t \left(\left[\sum_{i \in x_{ind}} (\frac{(XY[x] + t_{centroid})[i]}{XY_{sum}}) \cdot log(\frac{x_{sum} + t_{sum}}{(XY[x] + t_{centroid})[i]}) \right] + \frac{t_{sum}}{XY_{sum}} \cdot \left[\left[\sum_{i \in x_{ind}} \frac{t_{centroid}[i]}{t_{sum}} \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) \right] + log(\frac{x_{sum} + t_{sum}}{t_{sum}}) \right] \right) \tag{50}$$

$$= \arg\min_t \left(\frac{1}{XY_{sum}} \cdot \left(\left[\sum_{i \in x_{ind}} (XY[x] + t_{centroid})[i] \cdot log(\frac{x_{sum} + t_{sum}}{(XY[x] + t_{centroid})[i]}) \right] + t_{sum} \cdot \left[\left[\sum_{i \in x_{ind}} \frac{t_{centroid}[i]}{t_{sum}} \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) \right] + log(\frac{x_{sum} + t_{sum}}{t_{sum}}) \right] \right) \right) \tag{51}$$

Based on equation (19) :

$$= \arg\min_t \left(\left[\sum_{i \in x_{ind}} (XY[x] + t_{centroid})[i] \cdot log(\frac{x_{sum} + t_{sum}}{(XY[x] + t_{centroid})[i]}) \right] + t_{sum} \cdot \left[\left[\sum_{i \in x_{ind}} \frac{t_{centroid}[i]}{t_{sum}} \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) \right] + log(\frac{x_{sum} + t_{sum}}{t_{sum}}) \right] \right) \tag{52}$$

$$= \arg\min_t \left(\left[\sum_{i \in x_{ind}} (XY[x] + t_{centroid})[i] \cdot log(\frac{x_{sum} + t_{sum}}{(XY[x] + t_{centroid})[i]}) \right] + \left[\sum_{i \in x_{ind}} t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}} \right] + t_{sum} \cdot log(\frac{x_{sum} + t_{sum}}{t_{sum}}) \right) \tag{53}$$

Implementation Symbols

In the implementation we use the following symbols:

$$sum1 = \sum_{i \in x_{ind}} (XY[x] + t_{centroid})[i] \cdot log(\frac{x_{sum} + t_{sum}}{(XY[x] + t_{centroid})[i]}) \tag{54}$$

$$sum2 = \sum_{i \in x_{ind}} t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) \tag{55}$$

Computational Complexity

We get that computing the new centroid for a sample x is done in $O(|x_{ind}|)$ operations. Assuming that there are k centroids to select from, the overall complexity of computing $t^{new}(x)$ is $O(k \cdot |x_{ind}|)$

6.2 Computation for Dense Vector Representation

Let us assume that x is represented by a dense vector at the size of the vocabulary. In this representation, $x[i]$ indicates the frequency of the vocabulary item at index i in x.

Computation of $kl1$, $kl2$ and d_F

$$kl1 = KL(s, pi1 \cdot s + pi2 \cdot c) = \sum_i s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \tag{56}$$

$$kl2 = KL(c, pi1 \cdot s + pi2 \cdot c) = \sum_{i=0}^{d} c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \tag{57}$$

$$d_F(x,t) = (p(x) + p(t)) \cdot JS(p(y|x), p(y|t)) \tag{58}$$

$$= (p(x) + p(t)) \cdot JS(s, c) \tag{59}$$

$$= (p(x) + p(t))(pi1 \cdot kl1 + pi2 \cdot kl2) \tag{60}$$

$$= (p(x) + p(t))(\frac{p(x)}{p(x) + p(t)} \cdot kl1 + \frac{p(t)}{p(x) + p(t)} \cdot kl2) \tag{61}$$

$$= p(x) \cdot kl1 + p(t) \cdot kl2 \tag{62}$$

$$\begin{aligned} &= p(x) \cdot \left[\sum_i s[i] \cdot log(\frac{s[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] + \\ &\quad p(t) \cdot \left[\sum_i c[i] \cdot log(\frac{c[i]}{(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] \end{aligned} \tag{63}$$

$$\begin{aligned} &= p(x) \cdot \left[\sum_i s[i] \cdot \left[log(s[i]) + \frac{1}{log(pi1 \cdot s[i] + pi2 \cdot c[i])} \right] \right] + \\ &\quad p(t) \cdot \left[\sum_i c[i] \cdot \left[log(c[i]) + \frac{1}{log(pi1 \cdot s[i] + pi2 \cdot c[i])} \right] \right) \end{aligned} \tag{64}$$

$$\begin{aligned} &= \left[p(x) \cdot \sum_i s[i] \cdot log(s[i]) \right] + \left[p(t) \cdot \sum_i c[i] \cdot log(c[i]) \right] + \\ &\quad \left[\sum_i (p(x) \cdot s + p(t) \cdot c)[i] \cdot (\frac{1}{log(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] \end{aligned} \tag{65}$$

Computation of $t^{new}(x)$

$$\begin{aligned} t^{new}(x) &= \arg\min_t d_F(x,t) \\ &= \arg\min_t \left(\left[p(x) \cdot \sum_i s[i] \cdot log(s[i]) \right] + \left[p(t) \cdot \sum_i c[i] \cdot log(c[i]) \right] + \right. \\ &\qquad \left. \left[\sum_i (p(x) \cdot s + p(t) \cdot c)[i] \cdot (\frac{1}{log(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] \right) \end{aligned} \tag{66}$$

Based on equation (18) :

$$\begin{aligned} &= \arg\min_t \left(\left[p(t) \cdot \sum_i c[i] \cdot log(c[i]) \right] + \right. \\ &\qquad \left. \left[\sum_i (p(x) \cdot s + p(t) \cdot c)[i] \cdot (\frac{1}{log(pi1 \cdot s[i] + pi2 \cdot c[i])}) \right] \right) \end{aligned} \tag{67}$$

$$= \arg\min_t \left(\left[p(t) \cdot \sum_i p(y|t)[i] \cdot log(p(y|t)[i]) \right] - \left[\sum_i (p(x) \cdot p(y|x) + p(t) \cdot p(y|t))[i] \cdot log((pi1 \cdot p(y|x) + pi2 \cdot p(y|t))[i]) \right] \right) \tag{68}$$

We can now use equations (14) and (15), and the definition of $p(t)$ and $p(y|t)$ from Table 1:

$$= \arg\min_t \left(\left[\frac{t_{sum}}{XY_{sum}} \cdot \sum_i \frac{t_{centroid}[i]}{t_{sum}} \cdot log(\frac{t_{centroid}[i]}{t_{sum}}) \right] - \left[\sum_i (\frac{(XY[x] + t_{centroid})[i]}{XY_{sum}}) \cdot log(\frac{(XY[x] + t_{centroid})[i]}{x_{sum} + t_{sum}}) \right] \right) \tag{69}$$

$$= \arg\min_t \left(\frac{1}{XY_{sum}} \cdot \left(\left[\sum_i t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{t_{sum}}) \right] - \left[\sum_i (XY[x] + t_{centroid})[i] \cdot log(\frac{(XY[x] + t_{centroid})[i]}{x_{sum} + t_{sum}}) \right] \right) \right) \tag{70}$$

Based on equation (19) :

$$= \arg\min_t \left(\left[\sum_i t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{t_{sum}}) \right] - \left[\sum_i (XY[x] + t_{centroid})[i] \cdot log(\frac{(XY[x] + t_{centroid})[i]}{x_{sum} + t_{sum}}) \right] \right) \tag{71}$$

Implementation Symbols

In the implementation we use the following symbols:

$$sum1 = \sum_i t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{t_{sum}}) \tag{72}$$

$$sum2 = \sum_i (XY[x] + t_{centroid})[i] \cdot log(\frac{(XY[x] + t_{centroid})[i]}{x_{sum} + t_{sum}}) \tag{73}$$

Computational Complexity

We get that computing the new centroid for a sample x is done in $O(vocab\_size)$ operations. For k centroids, the complexity is $O(k \cdot vocab\_size)$

Data Structures

We showed two developments of the equations from the pseudo-code, for sparse and dense vectors representations. Let us examine the terms in these equations and explain the data structures that we use to store them:

- XY - the joint count matrix of the bag-of-words representation of the input. It is given as part of the inputs to the algorithm, either as a sparse matrix or a dense matrix.
- x_{sum} - the sum of values in the vector of sample x. It can be calculated once for each sample and stored for quick retrieval. We use an array of size $n\_samples$ to store the values of all samples.

- t_{sum} - the sum of values in the vectors of samples associated with cluster t. We use an array of size $n\_clusters$ to store this value for all clusters.
- $t_{centroid}$ - the sum of vectors of the samples associated with cluster t. We use a $2d$ dense matrix of size $n\_clusters \times vocab\_size$ to store all centroids.

7 Updating Clusters

Let us recall the workflow of partition optimization mentioned in Section (5) and repeated below:

1. Draw x_j out of $t(x_j)$
2. Compute $t^{new}(x_j) = \arg\min_{t'} d_(\{x_j\}, t')$
3. Merge x_j into $t^{new}(x_j)$

Let us now turn to the implementation of steps 1 and 3.

Drawing a sample x out of its cluster t is rather simple: we retrieve x's sum from the array holding all x_{sum} values, and we decrease it from t's entry in the array that maintains t_{sum}. This is a single operation of subtraction. In addition, we subtract x's vector values ($XY[x]$) from the centroid of t. If XY is a sparse matrix, this requires $|x_{ind}|$ operations. If XY is dense, it requires $vocab\_size$ operations.

Merging a sample into a new cluster is exactly the same but with additions instead of subtractions. Overall, updating clusters is significantly less compute-intensive than computing the new cluster for a sample.

8 Working with Uniform Prior

In its default mode, sIB clusters texts that are represented by count vectors, as generated by a Bag-of-Words model. However, this creates a bias towards longer texts, since such texts would have higher probability and influence the clustering analysis more than short texts.

To avoid this bias and assign all texts with equal probability, it is convenient to apply L_1 normalization to all vectors. This guarantees that all vectors sum to the same scalar (1.0) and have the same probability ($1/n\_samples$). This mode of operation is called *Uniform Prior*, while the default model is called *Native*.

The downside of using the *Uniform Prior* mode is that the vectors data type is no longer integer, which makes all computations more time-consuming and less convenient to optimize.

9 Caching *log* Computations

The most time-consuming operation in the computation of a new cluster for a sample is the *log* function. By examining some of the expressions in the equations, we can develop an optimization via caching.

In the case of sparse vector representations, we have:

$$sum2 = \sum_{i \in x_{ind}} t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) \tag{74}$$

Since x_{sum} and t_{sum} are scalars, their sum is a scalar, and we can separate its log computation from the log of $t_{centroid}$:

$$log(\frac{t_{centroid}[i]}{x_{sum} + t_{sum}}) = log(t_{centroid}[i]) - log(x_{sum} + t_{sum}) \tag{75}$$

In the case of dense vector representations, we have:

$$sum1 = \sum_{i} t_{centroid}[i] \cdot log(\frac{t_{centroid}[i]}{t_{sum}}) \tag{76}$$

And we can develop the log expression it in a similar way:

$$log(\frac{t_{centroid}[i]}{t_{sum}}) = log(t_{centroid}[i]) - log(t_{sum}) \quad (77)$$

We observe that $log(t_{centroid})$ is independent of the sample x for which we calculate the new cluster. Thus, we can use a matrix of the same size as $t_{centroid}$, to cache this computation and reuse it when we iterate over the samples. When a sample is drawn out of a cluster or merged into a cluster, we update only the entries in the cache matrix that refer to the cluster that has been updated.

The gain can be summarized as follows. Given a sample x for which we compute a new cluster, instead of computing the *log* operation over all centroids, we will compute the *log* only over two centroids – when updating the cached centroid of the cluster from where x is drawn out, and the one of the cluster to which x is merged into.

10 Experimental Optimizations

As we mentioned earlier, the *log* function is the most time-consuming operation in the computation of a new cluster for a sample. We have shown a way to optimize the computation of $log(t_{centroid})$ via simple caching and now we will discuss a way to optimize the computation of $log(XY[x] + t_{centroid})$, appearing both in $sum1$ of the development for sparse vector representation and in $sum2$ of the development for dense vector representation.

This optimization is based on replacing *log* invocation with a retrieval from lookup table in which $log(i)$ is pre-computed for every integer value in a defined range. This is an experimental direction that is still under investigation. If it will prove successful, it could replace the caching optimization proposed in Section (9).

In the *Native* mode, all expressions are integers, thus if a, b, c and d are integers, we can develop any expression of the form $log(\frac{a+b}{c+d})$ to $log(a+b) - log(c+d)$ and therefore *log* will be applied only to integers. Due to that, a lookup table (array) in which the *log* is pre-computed can be used to avoid the computation of *log* in run-time. First experiments in this approach indicated a substantial improvement in run-time. The smaller the data, the more likely that the full lookup table will fit into the CPU cache and be accessible more quickly.

In the *Uniform Prior* mode, a lookup table solution is less straightforward to use. Expressions such as $t_{centroid}$ are computed by summing a large number of rational numbers, each having a different denominator due to the different L_1 norm of every sample, thus requiring a very large common denominator for keeping a rational representation. However, even for a small set of samples, this denominator quickly exceeds any reasonable integer representation. To deal with this, either a floating-point or a fixed-point representation should be used, but any such representation has a penalty in precision when translated to an integer that can be used for accessing a lookup table.

References

[1] Ken Lang. Newsweeder: Learning to filter netnews. In *in Proceedings of the 12th International Machine Learning Conference (ML95)*, 1995.

[2] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 1982.

[3] Noam Slonim, Nir Friedman, and Naftali Tishby. Unsupervised document classification using sequential information maximization. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, page 129–136, New York, NY, USA, 2002. Association for Computing Machinery.