BCTH: A Novel Text Hashing Approach via Bayesian Clustering

Wenjie Ying[†,*], **Yuquan Le**[‡,*], **Hantao Xiong**[‡]
[†] Baidu Inc., Beijing, China
[‡] Changsha Lvzhidao Information Technology Co., Ltd., Changsha, China
yingwenjie@baidu.com, leyuquan@yeah.net, xionghantao94@163.com

Abstract

Similarity search is to find the most similar items for a certain target item. The ability of similarity search at large scale plays a significant role in many information retrieval applications and has received much attention. Text hashing is a promising strategy, which utilizes binary encoding to represent documents, and is able to obtain attractive performance. This paper makes the first attempt to utilize **B**ayesian **C**lustering for **T**ext **H**ashing, dubbed as BCTH. Specifically, BCTH can map documents to binary codes by utilizing multiple Bayesian Clusterings in parallel, where each Bayesian Clustering is responsible for one bit. Our approach employs the bit-balanced constraint to maximize the amount of information in each bit. Meanwhile, the bit-uncorrelated constraint is adopted to keep independence among all bits. The time complexity of BCTH is linear, where the hash codes and hash functions are jointly learned. Based on four widely-used datasets, the experimental results demonstrate that BCTH is competitive compared with currently competitive baselines from the perspective of both precision and training speed.

1 Introduction

The task of *similarity search*, also called *nearest neighbor search*, aims to find the most similar objects for a given query item (Gionis et al., 1999; Andoni and Indyk, 2006). It plays a significant role in many information retrieval applications, such as document clustering, content-based retrieval, collaborative filtering (Wang et al., 2016), etc. With the development of many intelligent terminals, massive textual data has been produced over the past several decades. Huge challenges exist in applying text similarity algorithms (Conneau et al., 2017; Le et al., 2018) to large-scale corpora, since these methods require complicated numerical computation.

Text hashing (Severyn and Moschitti, 2015) is a promising strategy and has obtained much attention. It maps semantically similar documents to hash codes with similar semantics through designing binary codes in a low-dimensional space. A hashing representation of each document usually needs only a few bits to be stored. The calculation of the similarity between two hash codes can be executed by a bit-wise XOR operation. Therefore, text hashing is an effective strategy to accelerate similarity queries and reduce data storage.

Most of the traditional text hashing methods consist of two stages (Zhang et al., 2010; Lin et al., 2014b; Severyn and Moschitti, 2015). The first step is to learn hash code, preserving similarity among neighbors. Then the hash function is trained through the self-taught method, with the text features and hash codes as the input (Wang et al., 2013b). However, for m documents, $O(m^2)$ training time complexity is needed to generate the pairwise similarity matrix used to preserve the similarity information. On the other hand, due to the success of deep learning, researchers have attempted to study text hashing through deep neural networks (Xu et al., 2015). Some of the most representative works include VDSH (Chaidaroon and Fang, 2017) and NASH (Kalchbrenner et al., 2014). The NASH model studies text hashing through an end-to-end Neural Architecture, which treats the hash codes as the latent factor. The VDSH model introduces a latent factor for documents to capture the semantic information. Even though these methods have achieved attractive performance, the training time is unsatisfactory, making them unscalable to large-scale datasets.

Motivated by the above observations, this paper attempts to utilize Bayesian Clustering for Text Hashing, dubbed as BCTH. Specifically, BCTH

[*] means the corresponding author.

Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, pages 54–62

can map documents to binary codes by using multiple Bayesian Clusterings in parallel, where each Bayesian Clustering is responsible for one bit. Our approach employs the bit-balanced constraint to maximize the amount of information in each bit. Meanwhile, the bit-uncorrelated constraint is adopted to keep independence among all bits. Experimental results prove that our approach is competitive in the perspective of both precision and training speed.

Our contributions are summarized as follows:

- We propose a novel Text Hashing based on the Bayesian Clustering framework, dubbed as BCTH, for learning effective hash codes from documents. To the best of our knowledge, this is the first work that utilizes Bayesian Clustering in text hashing.
- The time complexity of our method is linear, where the hash codes and hash function are jointly learned. What's more, we visualize the hash codes and prove that BCTH can obtain effective semantics from the original documents.
- We conduct extensive experiments on four public text datasets. Based on four widely-used datasets, the experimental results demonstrate that BCTH is competitive compared with currently competitive baselines from the perspective of both precision and training speed.

2 Model

The approach of our proposed BCTH is introduced in this section. As is shown in Fig. 1, BCTH is a general learning idea, which utilizes Bayesian Clustering that is based on the latent factor framework in Text Hashing. BCTH can map documents to binary codes by using multiple Bayesian Clusterings in parallel, where each Bayesian Clustering is responsible for one bit. During this process, the bit-balanced constraint is to maximize the amount of information in each bit. Meanwhile, the bit-uncorrelated constraint is adopted to keep independence among all bits.

2.1 Preliminaries

Given a set of m documents $\mathbf{X} = \{x^{(i)}\}_{i=1}^{m}$, where $x^{(i)}$ is the feature representation of the *i*-th document. The binary code for the *i*-th document is expressed as $b^{(i)} = \{b_k^{(i)}, b_k^{(i)} \in \{-1, 1\}\}_{k=1}^{r}$, and r is the length of the hash codes. Unlike the existing approaches (Liu et al., 2011; Zhang et al., 2010; Xu et al., 2015) that aim to preserve the pair-wise similarity among all the documents, we use Naive Bayes to extract the semantic information of the *i*-th document as:

$$P\left(b_k^{(i)} = c_k | x^{(i)}\right) = \frac{P\left(x^{(i)} | b_k^{(i)} = c_k\right) P\left(b_k^{(i)} = c_k\right)}{P\left(x^{(i)}\right)} \tag{1}$$

The Naive Bayes method assumes the conditional independence for the conditional probability distribution, and therefore, we obtain the following equation:

$$P\left(x^{(i)} | b_k^{(i)} = c_k\right) = \prod_{j=1}^{n} P(w_j = l_j^{(i)} | b_k^{(i)} = c_k) \tag{2}$$

where c_k represents the *k*-th bit's value of the hash codes of the *i*-th document, $c_k \in \{-1, 1\}$, and n is the size of the vocabulary. The $l_j^{(i)}$ denotes whether the *j*-th word of the vocabulary appears in the *i*-th document, and $l_j^{(i)} \in \{0, 1\}$.

The previous formula adopts the cumulative multiplication of all words' probabilities to calculate the likelihood of a particular document. However, since many words will not appear in a specific document, to avoid redundant calculation, we consider using the cumulative multiplication of the probabilities of words that appear in that particular document to calculate the probability of that document.

$$P\left(x^{(i)} | b_k^{(i)} = c_k\right) = \prod_{j \in \Phi^{(i)}} P(w_j = 1 | b_k^{(i)} = c_k) \tag{3}$$

In the above equation, $\Phi^{(i)}$ is a set of words that appear in the i-th document. Each hash code can be learned through an unsupervised iteration process. By utilizing multiple Bayesian Clusterings to calculate all hash codes of documents in parallel, we obtain the following objective function:

$$P\left(\mathbf{B} | \mathbf{X}\right) = \frac{P\left(\mathbf{X} | \mathbf{B}\right) P\left(\mathbf{B}\right)}{P\left(\mathbf{X}\right)} \tag{4}$$

where hash codes of documents are expressed as $\mathbf{B} = \{b_k^{(i)}, k = 1, 2..r, i = 1, 2.., m\}$.

In order to obtain high-quality hash codes, the bit-balanced and the bit-uncorrelated constraints are introduced. In addition, we transform the probability from the interval $[0, 1]$ to the interval $[-1, 1]$

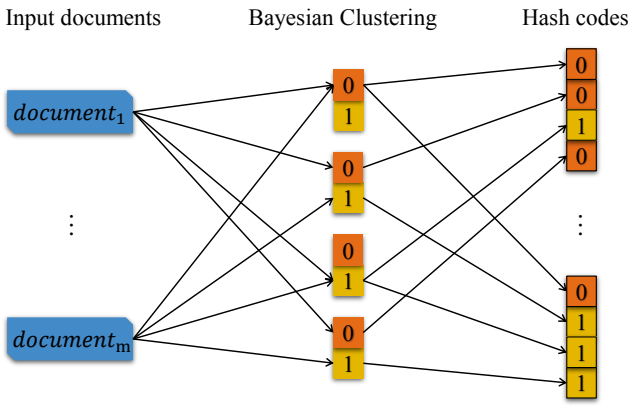

Figure 1: Illustration of how to learn the hash codes through multiple Bayesian Clusterings jointly from m documents. The size of hash codes in the illustration is $r = 4$.

by the function $f(P) = 2P - 1$. Therefore, we obtain the following loss function:

$$\begin{aligned} \min \sum_{k=1}^{r}\sum_{i=1}^{m} \left\| b_k^{(i)} - p_k^{(i)} \right\|^2 \iff \|\mathbf{B} - \mathbf{P}\| \\ s.t. \mathbf{B} \in \{-1, 1\}^{r \times m} \\ \mathbf{B1} = \mathbf{0}, \mathbf{B}\mathbf{B}^T = m\mathbf{I}_{r\times r} \end{aligned} \tag{5}$$

where $p_k^{(i)} = f\left(P(b_k^{(i)} = c_k | x^{(i)})\right)$ and $\mathbf{P} = \{p_k^{(i)}, k = 1, 2..r, i = 1, 2.., m\}$. The $\mathbf{1}$ denotes a vector with all of its elements equal to 1. The equality $\mathbf{B1} = \mathbf{0}$ denotes the bit-balanced constraint, which aims to maximize the amount of information in each bit. The equality $\mathbf{B}\mathbf{B}^T = m\mathbf{I}_{r\times r}$ denotes the bit-uncorrelated constraint, aiming to keep the independence among all bits.

However, the Eq. (5) is difficult to solve directly. Following the prior work in discrete graph hashing (Liu et al., 2014), let us define the constraint space as $\Omega = \left\{\mathbf{Y} \in \mathbf{R}^{r\times m} | \mathbf{Y1} = \mathbf{0}, \mathbf{Y}\mathbf{Y}^T = m\mathbf{I}_{r\times r}\right\}$. Then we formulate a more general framework which softens the two hard constraints in Eq. (5) as:

$$\begin{aligned} \min \|\mathbf{B} - \mathbf{P}\|^2 + \lambda \|\mathbf{B} - \mathbf{Y}\|^2 \\ s.t. \mathbf{B} \in \{-1, 1\}^{r \times m} \\ \mathbf{Y1} = \mathbf{0}, \mathbf{Y}\mathbf{Y}^T = m\mathbf{I}_{r\times r} \end{aligned} \tag{6}$$

where $\lambda \geq 0$ is a hyper parameter and Y is relaxation factor. If problem (5) is feasible, we can enforce $\mathbf{B1} = \mathbf{0}, \mathbf{B}\mathbf{B}^T = m\mathbf{I}_{r\times r}$ in Eq.(5) by setting an extremely large value to λ, thereby converting problem (6) into problem (5).

2.2 Learning

The learning process aims to find the desirable hash codes that can optimize the Eq. (6). Similar to (Liu et al., 2014), we utilize a tractable alternating minimization algorithm, which is an unsupervised iteration process, including alternately solving three sub-problems.

W-subproblem: Let us initialize the hash codes B randomly, and the parameter $W = \{p(w_j = 1|b_k = c_k), p(b_k = c_k)\}$, $j \in \{1, 2, ..., n\}$, $k \in \{1, 2, ..., r\}$ can be calculated by Naive Bayes. The document is represented by the one-hot method. The variable $\mathbf{P}$ is calculated in the following way, specifically, through the conditional probability and prior probability. The formula is as follow:

$$p(b_k = c_k) = \frac{\sum_{i=1}^{m} \boldsymbol{I}(b_k^{(i)} = c_k)}{m}, k \in \{1, 2, ..., r\} \tag{7}$$

where the $p(b_k = c_k)$ is the ratio of the number of documents with the k-th hash code equal to c_k to the total number of documents. I is the indicator function. If the input value is true, it returns 1, else returns 0. The calculation process of the conditional probability $P(w_j = 1|b_k = c_k)$, which includes the strategy of Laplace smoothing, is as follow:

$$P(w_j = 1|b_k = c_k) = \frac{\sum_{i=1}^{m} \boldsymbol{I}(w_j^{(i)} = 1 \cap b_k^{(i)} = c_k) + 1}{\sum_{j=1}^{n}\sum_{i=1}^{m} \boldsymbol{I}(w_j^{(i)} = 1 \cap b_k^{(i)} = c_k) + n} \tag{8}$$

where $\sum_{i=1}^{m} \boldsymbol{I}(w_j^{(i)} = 1 \cap b_k^{(i)} = c_k)$ is the number of documents whose k-th hash code value is c_k, which contains the word w_j. The " $\cap$ " symbol means "and".

Y-subproblem: Given the value of $\mathbf{B}$, the continuous variable $\mathbf{Y}$ can be calculated by Eq. (10),

the details are as follows:

$$\min_{\mathbf{Y}} \|\mathbf{B}-\mathbf{Y}\|^2 \iff \min_{\mathbf{Y}} 2(m-tr(\mathbf{B}^T\mathbf{Y}))$$
$$s.t. \mathbf{Y1}=\mathbf{0}, \mathbf{YY}^T=m\mathbf{I}_{r\times r} \tag{9}$$

Where tr is solving the trace of a matrix and $\mathbf{I}_{r\times r}$ is an identity matrix.

Minimizing $2(m-tr(\mathbf{B}^T\mathbf{Y}))$ is equivalent to maximizing the trace of the $\mathbf{B}^T\mathbf{Y}$, and it can be solved by performing singular value decomposition (SVD) operation on the matrix $\overline{\mathbf{B}}$ where every element is calculated by: $\overline{b}_k^{(i)} = b_k^{(i)} - \frac{1}{m}\sum_{i=1}^{m} b_k^{(i)}$. The $\mathbf{U}_b$ and $\mathbf{V}_b$, therein satisfying $[\mathbf{V}_b\ \mathbf{1}]^T\widehat{\mathbf{V}}_b = 0$, are stacked by the left and right singular vectors respectively from the result of SVD. After performing Gram-Schmidt process on $\mathbf{U}_b$ and $\mathbf{V}_b$, we obtain $\mathbf{U}_b$ and $\mathbf{V}_b$. Finally, according to (Zhang et al., 2016), the $\mathbf{Y}$ is updated by:

$$\mathbf{Y} = \sqrt{m}\left[\mathbf{U}_b\ \widehat{\mathbf{U}}_b\right]\left[\mathbf{V}_b\ \widehat{\mathbf{V}}_b\right]^T \tag{10}$$

B-subproblem: Given the value of $\mathbf{P}$ and the continuous variable $\mathbf{Y}$, the value of $\mathbf{B}$ can be calculated by minimizing Eq. (12), and the details are as follows:

$$\min_{\mathbf{B}} \|\mathbf{B}-\mathbf{P}\|^2 + \lambda\|\mathbf{B}-\mathbf{Y}\|^2$$
$$s.t. \mathbf{B} \in \{-1,1\}^{r\times m} \tag{11}$$

Since Eq. (12) is a simple binary optimization process, we can update $\mathbf{B}$ by updating each element of it in parallel according to:

$$\mathbf{b}_k^{(i)} = \underset{\mathbf{b}_k^{(i)}\in\{-1,1\}}{\operatorname{argmin}} \left\|\mathbf{b}_k^{(i)}-\mathbf{p}_k^{(i)}\right\|^2 + \lambda\left\|\mathbf{b}_k^{(i)}-\mathbf{y}_k^{(i)}\right\|^2 \tag{12}$$

The whole algorithm implementation process is shown in algorithm 1.

2.3 Complexity Analysis

In this section, we analyze the space and time complexity of BCTH. The learning algorithm of BCTH is shown in Algorithm 1. For space complexity, Algorithm 1 requires $O(mn+mr+nr)$ to store the training datasets, hash codes, and parameters. As r is usually less than 1024, we can easily store the above variables at large-scale in memory.

For time complexity, we first analyze each of the sub-problems. For $\mathbf{W}$-subproblem, it takes $O(mnr)$ to calculate parameter $\mathbf{W}$ and update probability $\mathbf{P}$. For $\mathbf{Y}$-subproblem, it requires $O(r^2n)$ to perform the SVD, Gram-Schmidt orthogonalization, and matrix multiplication. For $\mathbf{B}$-subproblem, it requires $O(mn)$ to update each $\mathbf{b}_k^{(i)}$ of $\mathbf{B}$. The time complexity of the whole Algorithm 1 is $O(t(mnr+r^2n+mn))$, where t is the number of iterations needed for convergence. In our experiments, t is set to 10 by default (See section 3.8). It can be seen that the time complexity of BCTH is linear.

Algorithm 1: Learning algorithm of BCTH

Input : Training data: $\mathbf{X} \in \mathbf{R}^{m\times n}$
code length: r
hyperparameter: λ;
Output : $W = \{p(w_j=1|b_k = c_k), p(b_k=c_k)\}, j \in \{1,2,...,n\}, k \in \{1,2,...,r\}$;

Initialize: $\mathbf{B}$ by randomization;
repeat
  W-step:
  Solve $\mathbf{W}$ and $\mathbf{P}$ in W-subproblem
  Y-step:
  Solve $\mathbf{Y}$ in Y-subproblem
  B-step:
  Solve $\mathbf{B}$ by B-subproblem
until *convergence*;
return $\mathbf{W}$,$\mathbf{B}$;

3 Experiments

3.1 Datasets

Following prior works (Chaidaroon and Fang, 2017), we experiment on four public text datasets.

- **Reuters Corpus Volume I (RCV1):** The *RCV1* is a large collection of manually labeled 800,000 newswire stories provided by *Reuters*. The full-topics version is available at the LIBSVM website[1].
- **Reuters21578 (Reuters)[2]:** This dataset is a widely-used text corpus for text classification. This collection contains 10,788 documents with 90 categories and 7,164 unique words.
- **TMC[3]:** This dataset has 22 labels, 21,519

[1] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multilabel.html
[2] http://www.nltk.org/book/ch02.html
[3] https://catalog.data.gov/dataset/siam-2007-text-mining-competition-dataset

training set, 3,498 test set, and 3,498 documents for the validation set. This dataset is used as part of the SIAM text mining competition and contains the air traffc reports provided by NASA.

- **20Newsgroups**[4]**:** The 20 Newsgroups dataset is a collection of 18828 newsgroup documents. It is divided into different newsgroups, each corresponding to a specific topic.

3.2 Baselines and Evaluation Metrics

We compare BCTH with the following competitive unsupervised methods since BCTH also belongs to unsupervised methods.

- **LSH:** This approach applies (Datar et al., 2004) random projections as the hash function to transform the data points from its original space to the binary hash space. More hash bits are needed to guarantee the precision on account of the randomness of the hash function.
- **SH:** This baseline (Weiss et al., 2008) calculates the bits through thresholding a subset of eigenvectors of the Laplacian of the similarity graph.
- **STH:** STH (Zhang et al., 2010) aims to find the best *l*-bit binary codes for all documents in the corpus via unsupervised learning.
- **AGH:** This method (Liu et al., 2011) discovers the neighborhood structure hidden in the data to learn proper compact codes. To make the method computationally feasible, it utilizes Anchor Graphs to gain tractable low-rank adjacency matrices.
- **VDSH:** (Chaidaroon and Fang, 2017) presents a series of deep learning models for text hashing, including VDSH, VDSH-S, and VDSH-SP. The VDSH-S and VDSH-SP models are supervised by utilizing document labels/tags for the hashing process. For the comparison's fairness, the VDSH is adopted as the baseline since our method is also unsupervised.

To better evaluate the effectiveness of hash codes used in the field of similarity search, every document in the test set is adopted as the query document. The similarity between the query document and each target similar document, which is utilized to retrieve relevant documents, is calculated by the Hamming distance of their hash codes respectively. The performance is measured by Precision, which is the ratio of the number of the similar documents to the number of total retrieved documents. The retrieved document that shares any common test label with the query document is denoted as a relevant document. Similar to previous works (Chaidaroon and Fang, 2017), the precision for the top 100 (pre@100) is employed as the main criterion. The final results are averaged over all the test documents.

3.3 Experimental Setup

We randomly split each dataset into two subsets for training and testing, which account for 90% and 10%, respectively. The training data is used to learn the mapping from the document to the hash code. Each document in the test set is used to retrieve similar documents based on the mapping, and the results are evaluated. The similar as (Chaidaroon and Fang, 2017), we use one-hot encoding as the default representation of the raw document. The hyper-parameter $\lambda = [0, \mathbf{0.001}, 0.01, 0.1]$, where the number in bold denotes the default setting (see Section 3.7). The number of iterations is set to 10 (see Section 3.8). Our codes are available at the open-source code repository [5].

In addition, the settings of the SH[6], AGH[7], STH[8] and VDSH[9] remain unchanged with original paper. We run five trials for each methods and an average of five trials is reported to avoid bias introduced by randomness. All of the methods are run on Windows with 1 Intel i7-7500 CPU and 1 GeForce GTX 1050Ti GPU.

3.4 Comparison Results

To examine the competitiveness of BCTH, we compared our method with competitive baselines, including traditional techniques and deep learning models from the perspective of both precision and training speed.

Table 2 reports the training time on the 20Newsgroups dataset. From the table, we can derive the following interesting conclusions: (1) Compared

[4]http://ana.cachopo.org/datasets-for-single-label-text-categorization

[5]https://github.com/myazi/SemHash

[6]https://github.com/superhans/SpectralHashing

[7]https://github.com/ColumbiaDVMM/Anchor-Graph-Hashing

[8]http://www.dcs.bbk.ac.uk/ dell/publications/dellzhang_sigir2010/sth_v1.zip

[9]https://github.com/unsuthee/VariationalDeepSemanticHashing

Table 1: Precision of the top 100 retrieved documents on four datasets with different numbers of hashing bits. The bold font denotes the best result at that number of bits.

Methods	RCV1					Reuters				
	8bits	**16bits**	**32bits**	**64bits**	**128bits**	**8bits**	**16bits**	**32bits**	**64bits**	**128bits**
LSH	0.4180	0.4352	0.4716	0.5214	0.5877	0.2802	0.3215	0.3862	0.4667	0.5194
SH	0.5321	0.5658	0.6786	0.7337	0.7064	0.4016	0.4201	0.4631	0.4590	0.4622
STH	0.6992	0.7688	0.8016	0.8098	0.8037	**0.6955**	0.7239	0.7576	0.7486	0.7240
AGH	0.4257	0.4976	0.5457	0.5698	0.5799	0.6552	0.7046	0.7313	0.7189	0.7043
VDSH	0.7285	0.7718	0.8165	0.7720	0.6630	0.6642	0.7118	0.7335	0.7083	0.7079
BCTH	**0.7339**	**0.7989**	**0.8389**	**0.8641**	**0.8690**	0.6827	**0.7307**	**0.7584**	**0.7669**	**0.7889**
Methods	**20Newsgroups**					**TMC**				
	8bits	**16bits**	**32bits**	**64bits**	**128bits**	**8bits**	**16bits**	**32bits**	**64bits**	**128bits**
LSH	0.0578	0.0597	0.0666	0.0770	0.0949	0.4388	0.4393	0.4514	0.4553	0.4773
SH	0.0699	0.1096	0.2010	0.2732	0.2632	0.5999	0.6206	0.6108	0.5813	0.5612
STH	0.2035	0.3481	0.4581	0.5129	0.5247	**0.7278**	**0.7520**	0.7633	0.7569	0.7411
AGH	0.2435	0.3531	0.3861	0.3796	0.3579	0.6000	0.6334	0.6443	0.6423	0.6273
VDSH	**0.3514**	0.3848	0.4667	0.2219	0.0651	0.6503	0.6640	0.7062	0.6567	0.5868
BCTH	0.3089	**0.4497**	**0.5216**	**0.5534**	**0.5830**	0.7076	0.7351	**0.7651**	**0.7804**	**0.7926**

Methods	8bits	16bits	32bits	64bits	128bits
SH	28.1	28.9	32.2	37.8	47.1
STH	16.3	16.5	17.9	20.3	28.3
AGH	10.3	10.8	11.4	12.8	15.5
VDSH	100+	100+	100+	100+	100+
BCTH	**0.5**	**0.9**	**2.0**	**5.0**	**10.8**

Table 2: Training time (second) of different methods on *20Newsgroups* dataset.

with these methods, BCTH costs less training time among all different hash bits. The reason can be attributed to the joint learning of hash codes and hash function, without needing to build the pairwise similarity matrix and the linear time complexity of BTCH. (2) It consumes extremely more time to train the VDSH model than train a traditional model. It shows that deep learning methods with sophisticated network architecture bring many parameters, thus requiring much more time to complete the training process. (3) The SH, STH, and AGH spend less time on the training process, which indicates that the traditional methods has its advantage over the deep learning method in training time.

Apart from comparing the 20Newsgroups dataset, we also compare over four datasets from the perspective of precision. Table 1 reports the comparison results with various methods over different numbers of bits. From this table, we can derive the following interesting conclusions: (1) Our proposed BCTH outperforms nearly all baselines among all different hash bits on four datasets. It demonstrates that BCTH, which introduces the bit-balanced and the bit-uncorrelated constraints, can learn effectively hash code from documents. (2) The VDSH model outperforms traditional methods in almost all situations. It denotes that deep learning techniques can capture inherent hidden text semantics, which are beneficial to generate the text hash codes. Although our method does not get the best results in some datasets under the circumstance of short hashing bit, it is approximating the best ones. Since our method utilizes the bit-balanced and the bit-uncorrelated constraints to make each bit capture independent semantics for documents, it is worthwhile to study the relationship between the length of the hash codes and the effect of our method.

3.5 Impact of the Length of Hash Codes

Previous works usually limit the length of the hash code to 128 bits on account of data storage. To study the effectiveness of the hash codes' size, we conduct experiments on hash codes ranging from 8 bits to 128 bits and extend hash codes to 1024 bits in this section.

Figure 2 reports the compared results on four datasets. From this figure, we can find the following phenomena: (1) when the length of the hash codes is equal to or over 128 bits, the effect of most other methods starts to decline. (2) the performance of our method always increases with the length of the hash codes increasing over all datasets. The

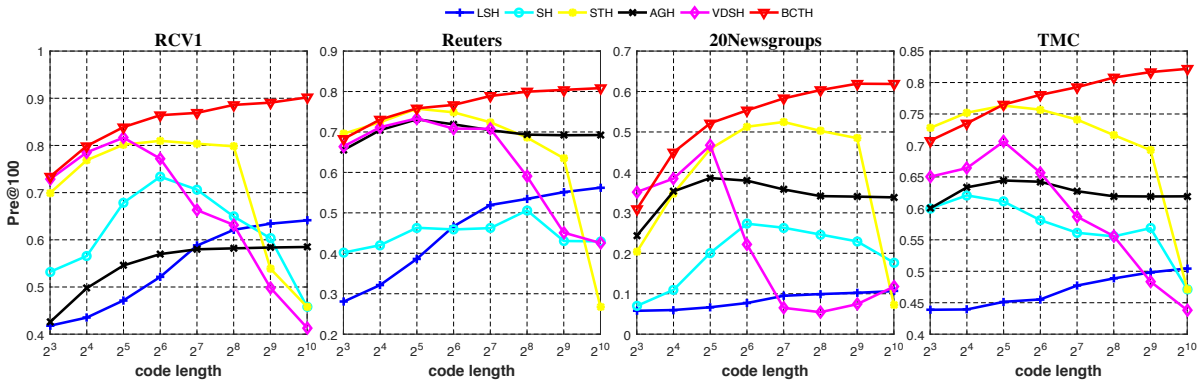

Figure 2: Precision@100 curve on four datasets with hash codes length from 8 to 1024.

reason is that our approach, which introduces the bit-balanced and the bit-uncorrelated constraints, can better keep independent semantic for all bits.

3.6 Qualitative Analysis

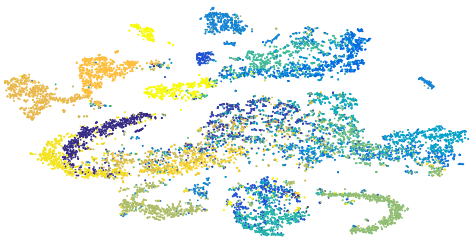

Figure 3: Visualization of the 1024-dimensional document latent semantic vectors by BCTH on the *20Newsgroup* dataset using t-SNE.

To evaluate whether our presented BCTH model can preserve the original documents' semantics, we visualize the documents' low-dimensional representations on the 20Newsgroups dataset in this section. In particular, the hash codes, obtained by BCTH, can be regarded as the latent semantic vectors of documents. We use t-SNE[10] tool to generate the scatter plots through 1024-bit hash codes. Figure 3 shows the results. Different colors represent different categories based on the ground truth. As we can see from figure 3, BCTH generates well-separated clusters with each corresponding to a true category. It shows that our method can effectively learn low-dimensional representations for documents.

3.7 Impact of Parameters

Our method is involved with a critical parameter λ, which is used to control the bit-balanced and the bit-uncorrelated constraints. We here study the impact of hyper-parameter λ in this section. Table 3 shows the results, which are obtained by using 64 hash bits. From this table, we can find that: (1) With λ varying from 0 to 0.1, BCTH is able to achieve relatively desirable results over all four datasets, which means that λ is universally applicable; (2) With λ set to 0.001, BCTH obtains the optimal result, and therefore, 0.001 is set as the default value for our method.

Datasets	$\lambda = 0$	$\lambda = 0.001$	$\lambda = 0.01$	$\lambda = 0.1$
RCV1	0.8476	**0.8641**	0.8526	0.8269
Reuters	0.7577	**0.7669**	0.7668	0.7532
20Newsgroups	0.5528	**0.5534**	0.5464	0.5502
TMC	0.7073	**0.7172**	0.7070	0.7025

Table 3: The effect of λ on four datasets with 64 hashing bits.

3.8 Convergence Speed

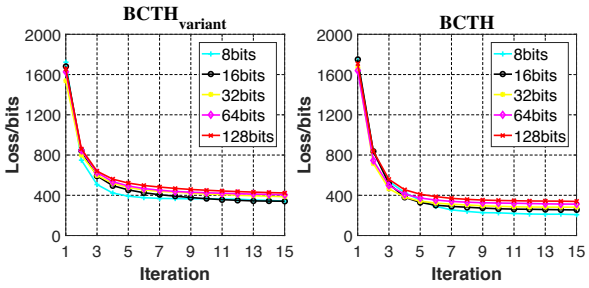

Figure 4: Convergence curve of the loss on the *20Newsgroups*.

In order to evaluate the convergence performance of our proposed BCTH algorithm, we performed convergence experiments on the *20Newsgroups* dataset. Considering the different loss scales produced under different hash bits, we consider the ratio of the loss to the hash length to make

[10]https://lvdmaaten.github.io/tsne/

it comparable at the same scale for different iterations. Note that, in this set of experiments, we also test one variant of BCTH methods, which is known as $\mathrm{BCTH}_{\mathrm{variant}}$, which calculates the conditional probability through Eq. (2). The result is reported in figure 4, and we can find that: (1) both the BCTH and the $\mathrm{BCTH}_{\mathrm{variant}}$ converge after approximately 10 iterations, and therefore, 10 is set as the default value for our method; (2) the convergence effect of BCTH is better than $\mathrm{BCTH}_{\mathrm{variant}}$, which has a lower loss. This demonstrates that BTCH is effective.

4 Related Work

Hashing methods can be divided into data-independent methods and data-dependent methods (Chang et al., 2012). The well-known data-independent methods include locality sensitive hashing (LSH) (Datar et al., 2004) and its variants. Data-dependent hashing methods are also known as learning to hash (L2H) methods by learning a hash function from data (Li et al., 2016). At present, the main L2H methods (Wang et al., 2018) can be divided into three categories: pairwise similarity preserving, multiwise similarity preserving, and implicit similarity preserving. The pairwise similarity-preserving methods aim to build a pairwise similarity matrix between two points, such as spectral hashing (SH) (Weiss et al., 2008), hashing with graphs (AGH) (Liu et al., 2011), discrete graph hashing (DGH) (Liu et al., 2014), fast supervised hashing (FastH) (Lin et al., 2014a) and column-sampling-based discrete supervised hashing (COSDISH) (Kang et al., 2016). The multiwise similarity-preserving is similar to pairwise similarity, which uses three or more samples as a group to define generalized similarity measures (Norouzi et al., 2012; Wang et al., 2013a). The implicit similarity-preserving methods maintain the similarity in an equivalent manner that adopts the idea of preserving the similarity of local neighbors (Irie et al., 2014). Compared with this line of works, although our work also focuses on the nearest neighbor search, our work is different from theirs since (1) most of these works focus on images data, and (2) Bayesian Clustering is not covered in these works.

Another line of works discuss text hashing, is related to our work since our work also aims to learn binary code from documents effectively. For example, (Zhang et al., 2010) presented the Self-Taught Hashing (STH) method for efficiently learning semantic hashing. (Zhang et al., 2010) incorporated both the tag information and the similarity information from probabilistic topic modeling. However, many of these models rely on pairwise similarity-preserving technique, which the time complexity is unavoidable $O(m^2)$ where m is the number of documents. On the other hand, researchers have attempted to study text hashing (Xu et al., 2015) via deep neural networks owing to the success of deep learning. For example, (Chaidaroon and Fang, 2017) introduces a latent factor for documents to capture the semantic information. (Kalchbrenner et al., 2014) proposed an end-to-end Neural Architecture for Semantic Hashing (NASH), which treats the hashing codes as latent variables. Compared to this line of works, our work shares several common features: (1) our work also learns hashing by introducing latent factor, and (2) our work also aims to the issues related to text hashing. Nevertheless, our work differs from theirs in several features: (1) most of these works are based on complex nonlinear functions like convolutional neural networks, and training time complexity is enormous, and (2) Bayesian Clustering is not covered in these works. In this paper, we make the first attempts to utilize Bayesian Clustering for text hashing and gain training time's linear complexity.

5 Conclusion

This paper presents a general learning framework that utilizes multiple Bayesian Clusterings jointly for text hashing. We introduce two constraints to make the hash code effectively. Specifically, the bit-balanced constraint is employed to maximize the amount of information in each bit, and the bit-uncorrelated constraint is adopted to keep the independence among all bits. The time complexity of our method is linear. Based on four widely-used datasets, the experiment results demonstrate that BCTH is competitive compared with current competitive baselines from the perspective of both precision and training speed.

References

Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE Symposium on Foundations of Computer Science*, pages 459–468.

Suthee Chaidaroon and Yi Fang. 2017. Variational deep semantic hashing for text documents. In *Pro-*

ceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 75–84.

ShihFu Chang, YuGang Jiang, Rongrong Ji, Jun Wang, and Wei Liu. 2012. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Twentieth Symposium on Computational Geometry*, pages 253–262.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of Vldb Conference*, volume 8, pages 518–529.

Go Irie, Zhenguo Li, Xiao Ming Wu, and Shih Fu Chang. 2014. Locally linear hashing for extracting non-linear manifolds. In *Computer Vision and Pattern Recognition*, pages 2123–2130.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *In Proceedings of the Association for Computational Linguistics*.

Wang Cheng Kang, Wu Jun Li, and Zhi Hua Zhou. 2016. Column sampling based discrete supervised hashing. In *AAAI Conference on Artificial Intelligence*, pages 1230–1236.

Yuquan Le, Zhi-Jie Wang, Zhe Quan, Jiawei He, and Bin Yao. 2018. Acv-tree: A new method for sentence similarity modeling. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4137–4143.

Wu Jun Li, Sheng Wang, and Wang Cheng Kang. 2016. Feature learning based deep supervised hashing with pairwise labels. In *International Joint Conference on Artificial Intelligence*, pages 1711–1717.

Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van Den Hengel, and David Suter. 2014a. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1978.

Guosheng Lin, Chunhua Shen, David Suter, and Anton Van Den Hengel. 2014b. A general two-step approach to learning-based hashing. In *IEEE International Conference on Computer Vision*, pages 2552–2559.

W. Liu, C. Mu, S. Kumar, and S. F. Chang. 2014. Discrete graph hashing. *Advances in Neural Information Processing Systems*, 4:3419–3427.

Wei Liu, Jun Wang, Sanjiv Kumar, and Shih Fu Chang. 2011. Hashing with graphs. In *Proc. International Conference on , Machine Learning June*, pages 1–8.

Mohammad Norouzi, David J Fleet, and Ruslan Salakhutdinov. 2012. Hamming distance metric learning. *Advances in Neural Information Processing Systems*, 2:1061–1069.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *The International ACM SIGIR Conference*, pages 373–382.

J. Wang, T. Zhang, J. Song, N Sebe, and H. T. Shen. 2018. A survey on learning to hash. *IEEE Transactions on Pattern Analysis Machine Intelligence*, PP(99):769–790.

Jianfeng Wang, Jingdong Wang, Nenghai Yu, and Shipeng Li. 2013a. Order preserving hashing for approximate nearest neighbor search. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 133–142.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih Fu Chang. 2016. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57.

Qifan Wang, Dan Zhang, and Luo Si. 2013b. Semantic hashing using tags and topic modeling. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 213–222.

Yair Weiss, Antonio Torralba, and Robert Fergus. 2008. Spectral hashing. In *International Conference on Neural Information Processing Systems*, pages 1753–1760.

Jiaming Xu, Peng Wang, Guanhua Tian, Bo Xu, Jun Zhao, Fangyuan Wang, and Hongwei Hao. 2015. Convolutional neural networks for text hashing. In *International Conference on Artificial Intelligence*, pages 1369–1375.

Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25.

Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat Seng Chua. 2016. Discrete collaborative filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 325–334.