Binary Coding based Label Distribution Learning

Ke Wang, Xin Geng*

MOE Key Laboratory of Computer Network and Information Integration,
School of Computer Science and Engineering,
Southeast University, Nanjing 210096, China
{k.wang, xgeng}@seu.edu.cn

Abstract

Label Distribution Learning (LDL) is a novel learning paradigm in machine learning, which assumes that an instance is labeled by a distribution over all labels, rather than labeled by a logic label or some logic labels. Thus, LDL can model the description degree of all possible labels to an instance. Although many LDL methods have been put forward to deal with different application tasks, most existing methods suffer from the scalability issue. In this paper, a scalable LDL framework named Binary Coding based Label Distribution Learning (BC-LDL) is proposed for large-scale LDL. The proposed framework includes two parts, i.e., binary coding and label distribution generation. In the binary coding part, the learning objective is to generate the optimal binary codes for the instances. We integrate the label distribution information of the instances into a binary coding procedure, leading to high-quality binary codes. In the label distribution generation part, given an instance, the \( k \) nearest training instances in the Hamming space are searched and the mean of the label distributions of all the neighboring instances is calculated as the predicted label distribution. Experiments on five benchmark datasets validate the superiority of BC-LDL over several state-of-the-art LDL methods.

1 Introduction

Learning with ambiguity plays a fundamental role in many areas including machine learning, computer vision, data mining and so on. Although different varieties of learning paradigms have been proposed for learning with ambiguity, label distribution learning (LDL) has become one of the latest approaches to this goal because it can perform better than other paradigms in some real applications [Geng et al., 2013; Zhou et al., 2016; Xing et al., 2016; Yang et al., 2016; Gao et al., 2017]. Generally speaking, LDL assumes that an instance \( x \) is labeled by a real number \( d_{xy} \) to every possible label \( y \), representing the degree to which the corresponding \( y \) describes \( x \). Thus, \( d_{xy} \) is named the description degree of \( y \) to the instance \( x \). The vector consisting of the description degrees of all the labels is called label distribution because it is similar to probability distribution. Label distribution can not only model the ambiguity of “what describes the instance”, but also deal with the more general ambiguity of “how to describe the instance” [Xing et al., 2016]. Without loss of generality, LDL assumes that \( d_{xy} \in [0, 1] \) and \( \sum_y d_{xy} = 1 \). Different from other learning paradigms, single-label learning (SLL) and multi-label learning (MLL) for example, the main purpose of LDL is to learn a group of mapping functions from the instance to its label distribution.

In recent years, a collection of LDL algorithms have been proposed. The representative methods include IISLDL [Geng et al., 2013], conditional probability neural network (CPNN) [Geng et al., 2013], label distribution support vector regressor (LDSVR) [Geng and Hou, 2015], BFGS-LDL [Geng, 2016], PT-Bayes [Geng, 2016], PT-SVM [Geng, 2016], AA-BP [Geng, 2016], LDLogitBoost [Xing et al., 2016], AOSO-LDLogitBoost [Xing et al., 2016], sparsity conditional energy label distribution learning (SCE-LDL) [Yang et al., 2016], deep label distribution learning (DDL) [Gao et al., 2017] and so on. However, most of them suffer from one key limitation: as the size of training set increases, the training time grows rapidly.

To avoid the high training time consumption, Geng (2016) proposed a method known as AA-\( k \)-NN, which is an adaptation algorithm of \( k \)-Nearest Neighbor (\( k \)-NN) to solve the LDL problem. Compared with other LDL algorithms, the advantage of AA-\( k \)-NN is that it does not need a training process. However, AA-\( k \)-NN suffers from two drawbacks. Firstly, its performance heavily depends on the quality of the original features it uses, which are usually extracted in an unsupervised way. Thus, when the high quality features cannot be obtained, it may lead to poor performance. Secondly, when the size of dataset is large and the dimension of real-valued feature is high, the testing phase (linear search) will cost a great deal of time. Therefore, like most of other LDL methods, AA-\( k \)-NN is not applicable to large-scale LDL.

To address the above-mentioned problems, we employ the binary coding/hashing techniques to tackle the task of large-scale LDL, which is widely used for approximate nearest neighbor (ANN) search [Indyk and Motwani, 1998]. The purpose of hashing is to transform high-dimensional original features into compact binary codes. Since the representations

*Corresponding author.
of instances are compressed as short binary codes, the time
and memory cost of searching process can be significantly
reduced.

Existing hashing methods can be roughly categorized in-
to two groups, i.e., unsupervised hashing and supervised (in-
cluding semi-supervised) hashing. As label information is
useful for generating more discriminative hash codes, su-
upervised hashing methods can often achieve better perfor-
mance. Existing representative supervised hashing algo-
rithms include sequential projection learning for hashing (S-
PLH) [Wang et al., 2010], supervised hashing with kernel-
s (KSH) [Liu et al., 2012], two-step hashing (TSH) [Lin et
al., 2013], FastHash [Lin et al., 2014], supervised discrete
hashing (SDH) [Shen et al., 2015], column sampling based
discrete supervised hashing (COSDISH) [Kang et al., 2016],
supervised quantization (SQ) [Wang et al., 2016]. However,
most existing supervised hashing methods are developed for
the instances labeled by a logic label or some logic labels.
They cannot directly integrate the label distribution informa-
tion into the binary coding procedure. Thus, it is not much
appropriate to extend most existing supervised hashing meth-
ods to deal with LDL problem.

It is worth noting that Jiang and Li (2015) put forward
the scalable graph hashing (SGH) method, which exploits a
feature transformation technique to approximate the whole
graph. The time complexity of SGH is $O(n)$, because the
pairwise similarity graph matrix is avoided explicitly com-
puting. Although the performance of SGH is promising, the
binary codes are learned in an unsupervised manner and the
supervised information is not fully considered.

Inspired by AA-kNN and SGH, a scalable LDL frame-
work, termed Binary Coding based Label Distribution Learn-
ing (BC-LDL), is put forward to solve the large-scale LDL
problem. The main contributions of this paper are briefly list-
ed as follows:

- We propose a novel supervised binary coding method
  for the instances labeled by the label distributions. Be-
due to the correlations between the instances based on their semantic label distributions, highly
discriminatory binary codes can be generated by the pro-
posed method.

- Combining binary coding and ANN search together, we
  further propose a scalable LDL framework for efficiently
  solving the large-scale LDL problem. Compared with
  other NN-based LDL methods (e.g., AA-kNN), the time
cost of the testing phase can be greatly reduced, be-
cause the similarity of two instances can be rapidly mea-
sured by calculating the Hamming distance between two
groups of binary codes via XOR operations.

- Experimental results on five real-world datasets demon-
strate that the proposed BC-LDL is competitive with the
state-of-the-art LDL methods.

The rest of this paper is organized as follows. Firstly, the
details of the proposed BC-LDL method are presented. Sec-
ondly, the experimental results on five public datasets are re-
ported. Finally, the conclusion is given.

2 BC-LDL

2.1 Problem Formulation

Let $X = [x_1; x_2; \ldots; x_n] \in \mathbb{R}^{n \times m}$ denote the input s-
pace, $Y = \{y_1, y_2, \ldots, y_c\}$ denote the complete set of label-
s, and $d^y_i$ denote the description degree of the label $y_i \in Y$
to the instance $x \in X$. $c$ is the number of the labels.
Depending on the instance, $d^y_i$ needs to satisfy two contraints, i.e., $d^y_i \in \{0, 1\}$ and
$\sum_{y_i} d^y_i = 1$. Without loss of generality, let $X$ be normalized
to have zero mean. Suppose that we have a training set $Z = \{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)\}$, where $n$ is the number of instances, $x_i \in \mathbb{R}^{1 \times m}$, and $d_i = [d^y_{i1}, d^y_{i2}, \ldots, d^y_{in}]$ is the corresponding label distribution of the instance $x_i$.

Different from most existing LDL methods, BC-LDL does
not learn a group of mapping functions from $x \in \mathbb{R}^{1 \times m}$
to $d \in \mathbb{R}^{1 \times c}$ directly. BC-LDL aims at learning a group
of binary coding functions based on the training set $Z$, i.e.,
$h(x): \mathbb{R}^{1 \times m} \rightarrow \{-1, 1\}^{1 \times l}$, where $l$ is the length of bi-
ary codes. Then, the ANN search strategy is employed to
generate the label distributions for the testing instances. In
this paper, $h(x)$ is defined as follows:

$$h(x) = \text{sign}(xW + e),$$

where $\text{sign}(\cdot)$ denotes the element-wise sign function, which returns 1 if the element is a positive number and returns -1 otherwise. $W \in \mathbb{R}^{m \times l}$ is the mapping matrix, and $e \in \mathbb{R}^{1 \times l}$ is the bias vector.

2.2 Binary Coding

Similar to some existing supervised hashing methods [Liu et
al., 2012; Zhang and Li, 2014; Lin et al., 2014; Kang et al.,
2016], we expect to approximate the similarity matrix $S$ by
the generated binary codes of training instances. Thus, we have the following objective function:

$$\min_{B \in \{-1, 1\}^{1 \times l}} \|BS - BB^T\|_F^2,$$

where $B$ is the binary code matrix, and $\| \cdot \|_F$ denotes the Frobenius norm.

The analytical form of the objective function is:

$$\min_{W,e} \|S - \text{sign}(XW + E)\text{sign}(XW + E)^T\|_F^2,$$

where $E = [e, e, \ldots, e] \in \mathbb{R}^{n \times l}$.

2.3 Similarity Matrix Construction

As aforementioned, each instance is labeled by a distribu-
tion over all labels in LDL, rather than labeled by a logic
label or some logic labels. Thus, we cannot construct the similarity matrix by modeling the label consistency between
instances, which is used in many supervised hashing algo-
rithms [Wang et al., 2010; Liu et al., 2012; Lin et al., 2013;
Zhu and Gao, 2017]. Furthermore, we think a real number
within the range of -1 and 1 is more appropriate to represent
the similarity between two instances than a logic number (i.e.,
-1 or 1) in LDL, because the former is more discriminative.
Based on the above considerations, we adopt the adjusted cosine similarity (ACS) metric [Sarwar et al., 2001] to construct the similarity matrix \( S \in [-1, 1]^{n \times n} \). Given an instance pair \((x_i, x_j)\) from the training set, the semantic affinity of this instance pair is defined as follows:

\[
S_{ij} = \frac{(d_i - \mu)(d_j - \mu)^T}{||d_i - \mu||_2 ||d_j - \mu||_2} (4)
\]

where \(\cdot\cdot\cdot\) denotes the inner product of two vectors, \( || \cdot ||_2 \) denotes the \(L_2\) norm of the vector, \(\mu\) is the mean of the label distributions of all the training instances, and \(D_{is} = \frac{d_i - \mu}{||d_i - \mu||_2}\).

Then we can obtain the similarity matrix \( S = DD^T \). In fact, if we construct the similarity matrix directly, it will lead to the high time complexity and memory requirement. Therefore, we need to avoid the construction of similarity matrix \( S \) in our algorithm. In the following subsection, we will describe how to achieve it in detail.

### 2.4 Optimization

There is a matrix variable \( W \) and a vector variable \( e \) in our original objective function. Actually, if we denote \( \overline{X} = [X, 1_n] \) and \( \overline{W} = [W; e] \), we can omit \( e \) and obtain the following form with only a matrix variable \( W \):

\[
\min_{\overline{W}} ||lS - sign(\overline{X} \overline{W})sign(\overline{X} \overline{W})^T||_F. (5)
\]

The objective function in (5) is hard to be solved directly due to the existence of sign function. A common solution is to adopt the spectral relaxation trick [Weiss et al., 2009] and impose orthogonality constraints, i.e., \( W^T X^T X W = I_n \). Here, \( I_n \) denotes a \( n \times n \) identity matrix. Then, the optimization problem can be easily transferred into a generalized eigenvalue problem. However, it has been verified that non-orthogonal mapping functions can perform better than orthogonal ones in some real applications [Wang et al., 2010; Zhang and Li, 2014]. Thus, we apply a sequential learning strategy to learn the binary coding functions without imposing orthogonality constraints [Liu et al., 2012; Zhang and Li, 2014; Jiang and Li, 2015].

Suppose that we have learned the mapping functions \( \overline{w}_1, \overline{w}_2, \ldots, \overline{w}_{t-1} \), we need to learn the next mapping function \( \overline{w}_t \). We define a residual matrix \( R_t \) as follows:

\[
R_t = lS - \sum_{i=1}^{t-1} sign(\overline{X} \overline{w}_i)sign(\overline{X} \overline{w}_i)^T. (6)
\]

Following the objective function in (5), the corresponding optimization problem to learn \( \overline{w}_t \) can be defined as:

\[
\min_{\overline{w}_t} ||R_t - sign(\overline{X} \overline{w}_t)sign(\overline{X} \overline{w}_t)^T||_F. (7)
\]

By some algebraic calculation, we further rewrite the objective function in (7) as:

\[
||R_t - sign(\overline{X} \overline{w}_t)sign(\overline{X} \overline{w}_t)^T||_F^2
= tr[(R_t - A_t A_t^T)(R_t - A_t A_t^T)^T]
= tr(A_t A_t^T A_t A_t^T) - 2A_t^T R_t A_t + tr(R_t R_t^T) (8)
= n^2 - 2A_t^T R_t A_t + tr(R_t R_t^T)
= -2A_t^T R_t A_t + const,
\]

where \( A_t = sign(\overline{X} \overline{w}_t) \), and \( tr(\cdot) \) denotes the trace operator.

Thus, the objective function in (7) can be reformulated as:

\[
\max_{\overline{w}_t} sign(\overline{X} \overline{w}_t)^T R_t sign(\overline{X} \overline{w}_t). (9)
\]

The objective function in (9) is still hard to be solved directly due to the existence of sign function. Here we use a widely-adopted approximate representation of sign function [Wang et al., 2010; 2012], i.e., \( sign(a) \approx a \), to handle it. We rewrite it as follows:

\[
\max_{\overline{w}_t} \overline{w}_t^T X^T R_t X \overline{w}_t \quad \text{s.t.} \overline{w}_t^T \overline{w}_t = 1, (10)
\]

where the orthogonality constraint is set to avoid the trivial solution.

It is easy to see that the optimization problem in (10) is an eigensystem problem and the eigenvector of \( X^T R_t X \) corresponding to its largest eigenvalue \( \lambda_{max} \) is the approximate solution \( \overline{w}_t \):

\[
X^T R_t X \overline{w}_t = \lambda_{max} \overline{w}_t. (11)
\]

We define \( G_t = X^T R_t X \), then we can obtain:

\[
G_t = G_1 - \sum_{i=1}^{t-1} (X^T A_i)(X^T A_i)^T
= X^T lS X - \sum_{i=1}^{t-1} (X^T A_i)(X^T A_i)^T (12)
= l(X^T D)(X^T D)^T - \sum_{i=1}^{t-1} (X^T A_i)(X^T A_i)^T,
\]

where \( A_i = sign(\overline{X} \overline{w}_t) \). According to the equation (12), we can find that the construction of similarity matrix \( S \) is avoided during the optimization procedure. It considerably boosts the scalability of our algorithm.

### 2.5 Label Distribution Generation

Our BC-DLD adopts a two-step strategy to learn the label distributions for the testing instances. In the first step, BC-DLD generates the binary codes for a new testing instance \( \overline{x} \) according to the equation (1): \( b = sign(\overline{x} W + e) \). In the second step, ANN search is conducted to find the \( k \) nearest training instances represented by binary codes in the Hamming space. Then, the mean of the label distributions of all the searched training instances is calculated as the learned label distribution.
Algorithm 1 BC-LDL: Train Algorithm
1: **Input:** Feature matrix $X$; Adjusted label distributions matrix $D$; Code length $l$.
2: **Output:** Projection matrix $W$; Bias vector $e$.
3: $X \leftarrow [X, 1_n]$. 
4: $G_0 \leftarrow (X^T D)(X^T D)^T$.
5: $G_1 \leftarrow lG_0$. 
6: for $t = 1, \ldots, l$ do 
7: solve the following eigensystem problem $G_t \omega = \lambda_{max} \omega$; 
8: obtain the mapping vector $\omega_t$ corresponding to the largest eigenvalue $\lambda_{max}$.
9: $Z \leftarrow [X^T \text{sign} (X \omega_1)] [X^T \text{sign} (X \omega_1)]^T$.
10: $G_{t+1} \leftarrow G_t - Z$.
11: end for 
12: $W \leftarrow [\omega_1, \omega_2, \ldots, \omega_l]$.
13: $W \leftarrow W(1:\text{end}-1,:); e \leftarrow W(\text{end},:)$.

Algorithm 2 BC-LDL: Test Algorithm
1: **Input:** Test instance $\tilde{x}$; No. of nearest neighbors $k$.
2: **Output:** Label distribution $d$.
3: $\tilde{b} \leftarrow \text{sign}(\tilde{x} W + e)$.
4: $N_k \leftarrow$ the $k$ nearest neighbors of $\tilde{b}$ in the Hamming space.
5: $d \leftarrow$ the mean of the label distributions of the instances $\in N_k$.

2.6 Summary and Analysis
We summarize the proposed BC-LDL method in Algorithm 1 and Algorithm 2. In the training phase, the time complexity of BC-LDL includes two aspects, i.e., the initialization procedure and the sequential learning procedure. Firstly, initialization of $G_0$ will cost $O(cn(m+1) + c(m+1)^2)$. Secondly, the time cost of the sequential learning procedure is $O(l((m+1)^3+(m+1)n+(m+1)^2))$. The training time complexity of BC-LDL is linear to $n$, because $c, l, m$ is usually much less than $n$.

In the testing phase, the time cost of generating the binary codes for a testing instance is $O(lm)$ and searching the $k$ nearest training instances is $O(kn)$. Hence, the testing time complexity is $O(lm + kn)$ in total.

We can see that when the size of training set is large, the testing phase will cost much time. An effective solution for this problem is that we can construct a hash table to store the mean of the label distributions of these training instances with the same binary codes. Then, the mean is used as the predicted label distribution for a testing instance that have the same binary codes. The time cost of constructing a hash table is $O(2^l n)$ and predicting the label distribution for a testing instance will cost $O(lm + l2^l)$. We set $l = 8$ in this paper, and thus $2^l \ll n$. Altogether, the training time complexity is still $O(n)$. This method is named as BCTable.

### 3 Experiment
In this section, we evaluate the proposed algorithms on five different datasets: s-BU_3DFE (scores-Binghamton University 3D Facial Expression) [Zhou et al., 2015], COPM (Crowd Opinion Prediction on Movies) [Geng and Hou, 2015], Twitter_LDL [Yang et al., 2017], Ren-CECps [Quan and Ren, 2010] and MORPH [Ricanek and Tesafaye, 2006]. Due to the space limitation, here we only present the brief statistics of these evaluated datasets in Table 1. All the experiments are carried on a PC with Intel (R) Core (TM) CPU i5-6300@2.30GHz and 12GB RAM.

### 3.1 Experimental Settings
To validate the proposed methods, we comprehensively compare them with several baselines. These algorithms include four specialized algorithms IIS-LDL, BFGS-LDL, CPNN, LDSVR, and one adaptation algorithm AA-k-NN.

As suggested in [Geng, 2016], six measures are employed to evaluate the accuracy performance of all the methods. Among them, Chebyshev, Clark, Canberra, Kullback-Leibler (K-L), measure the distance between two distributions, thus they are the smaller the better. Cosine and Intersection are used to measure the similarity between two distributions, so they are the larger the better.

The parameters of all the methods used in the experiments are carefully tuned and the best results of baseline methods are reported. On s-BU_3DFE, $k$ in BC-LDL is set to 20 and code length is set to 32 bits. The maximum iteration steps in BFGS-LDL is 300 and IIS-LDL is 100. $k$ for AA-kNN is set to 20. On COPM, $k$ in BC-LDL is 30 and code length is 128 bits. The maximum iteration steps in BFGS-LDL is 100 and IIS-LDL is 20. $k$ in AA-kNN is 10. On Twitter_LDL, $k$ in BC-LDL is set to 10 and code length is 256 bits. The maximum iteration steps in BFGS-LDL is 300 and IIS-LDL is 50. $k$ in AA-kNN is 10. On Ren-CECps, $k$ in BC-LDL is 20 and code length is set to 64 bits. The maximum iteration steps in BFGS-LDL is 200 and IIS-LDL is 100. $k$ in AA-kNN is set to 20. On MORPH, $k$ in BC-LDL is set to 50 and code length is 256 bits. The maximum iteration steps in BFGS-LDL is 100 and IIS-LDL is 20. $k$ in AA-kNN is set to 10. On the five datasets, the insensitivity parameter $\varepsilon$ of LDSVR is set to 0.1 and the number of hidden-layer neurons of CPNN is set to 50.

All the results are averaged over 10-fold cross validation in terms of both accuracy and time cost.

### 3.2 Experimental Results
Table 2-6 report the results of our methods and other baselines on the five evaluated datasets. When we evaluate the performance of a LDL algorithm, its accuracy performance and efficiency performance are equally important. Hence, three different rank metrics are adopted to evaluate the performance of all the algorithms in our experiments, i.e., Accuracy Rank (Acc. Rank), Time Rank and Average Rank (Avg. Rank), which can reflect the accuracy, efficiency and comprehensive performance of the algorithms.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#instance</th>
<th>#dim</th>
<th>#label</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-BU_3DFE</td>
<td>2,500</td>
<td>243</td>
<td>6</td>
</tr>
<tr>
<td>COPM</td>
<td>7,755</td>
<td>1,869</td>
<td>5</td>
</tr>
<tr>
<td>Twitter_LDL</td>
<td>10,045</td>
<td>200</td>
<td>8</td>
</tr>
<tr>
<td>Ren-CECps</td>
<td>35,096</td>
<td>100</td>
<td>8</td>
</tr>
<tr>
<td>MORPH</td>
<td>55,132</td>
<td>200</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 1: Statistics of the five evaluated datasets.
Table 2: The experimental results on s-BU_{3DFE}. Items in bold indicate the best performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Cost (s)</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC-LDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.102(1)</td>
<td>0.323(1)</td>
<td>0.664(1)</td>
<td>0.054(1)</td>
</tr>
<tr>
<td>BCTable</td>
<td>0.117(4)</td>
<td>0.384(4)</td>
<td>0.763(4)</td>
<td>0.067(5)</td>
</tr>
<tr>
<td>BFGS-LDL</td>
<td>0.105(2)</td>
<td>0.355(2)</td>
<td>0.742(3)</td>
<td>0.056(3)</td>
</tr>
<tr>
<td>IIS-LDL</td>
<td>0.121(5)</td>
<td>0.379(5)</td>
<td>0.805(5)</td>
<td>0.066(4)</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.106(3)</td>
<td>0.346(2)</td>
<td>0.729(2)</td>
<td>0.055(2)</td>
</tr>
<tr>
<td>LDSVR</td>
<td>0.126(3)</td>
<td>0.383(6)</td>
<td>0.826(7)</td>
<td>0.072(6)</td>
</tr>
<tr>
<td>AA-ANN</td>
<td>0.125(6)</td>
<td>0.388(7)</td>
<td>0.826(6)</td>
<td>0.073(7)</td>
</tr>
</tbody>
</table>

Table 3: The experimental results on COPM. Items in bold indicate the best performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Cost (s)</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC-LDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.115(2)</td>
<td>0.518(2)</td>
<td>0.992(2)</td>
<td>0.102(2)</td>
</tr>
<tr>
<td>BCTable</td>
<td>0.118(3)</td>
<td>0.532(2)</td>
<td>1.017(4)</td>
<td>0.101(3)</td>
</tr>
<tr>
<td>BFGS-LDL</td>
<td>0.119(4)</td>
<td>0.522(3)</td>
<td>1.003(3)</td>
<td>0.100(4)</td>
</tr>
<tr>
<td>IIS-LDL</td>
<td>0.138(6)</td>
<td>0.561(6)</td>
<td>1.078(6)</td>
<td>0.122(6)</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.146(7)</td>
<td>0.615(7)</td>
<td>1.188(7)</td>
<td>0.157(7)</td>
</tr>
<tr>
<td>LDSVR</td>
<td>0.112(1)</td>
<td>0.499(1)</td>
<td>0.966(1)</td>
<td>0.097(1)</td>
</tr>
<tr>
<td>AA-ANN</td>
<td>0.121(5)</td>
<td>0.545(5)</td>
<td>1.041(5)</td>
<td>0.100(5)</td>
</tr>
</tbody>
</table>

Table 4: The experimental results on Twitter_{LDL}. Items in bold indicate the best performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Cost (s)</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC-LDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.266(2)</td>
<td>2.089(2)</td>
<td>4.841(1)</td>
<td>0.469(2)</td>
</tr>
<tr>
<td>BCTable</td>
<td>0.282(4)</td>
<td>2.396(3)</td>
<td>6.210(3)</td>
<td>0.584(4)</td>
</tr>
<tr>
<td>BFGS-LDL</td>
<td>0.466(6)</td>
<td>2.410(5)</td>
<td>6.289(5)</td>
<td>1.142(6)</td>
</tr>
<tr>
<td>IIS-LDL</td>
<td>0.500(7)</td>
<td>2.400(4)</td>
<td>6.249(4)</td>
<td>1.172(7)</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.306(5)</td>
<td>2.447(7)</td>
<td>6.421(7)</td>
<td>0.754(5)</td>
</tr>
<tr>
<td>LDSVR</td>
<td>0.267(3)</td>
<td>2.420(3)</td>
<td>6.338(6)</td>
<td>0.553(3)</td>
</tr>
<tr>
<td>AA-ANN</td>
<td>0.260(1)</td>
<td>2.090(2)</td>
<td>4.878(2)</td>
<td>0.465(1)</td>
</tr>
</tbody>
</table>

Table 5: The experimental results on Ren-CECPs. Items in bold indicate the best performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Cost (s)</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC-LDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.566(1)</td>
<td>2.329(1)</td>
<td>5.773(1)</td>
<td>1.229(1)</td>
</tr>
<tr>
<td>BCTable</td>
<td>0.600(3)</td>
<td>2.639(3)</td>
<td>7.261(3)</td>
<td>1.422(3)</td>
</tr>
<tr>
<td>BFGS-LDL</td>
<td>0.604(4)</td>
<td>2.649(5)</td>
<td>7.319(4)</td>
<td>1.490(5)</td>
</tr>
<tr>
<td>IIS-LDL</td>
<td>0.625(6)</td>
<td>2.654(6)</td>
<td>7.357(6)</td>
<td>1.564(6)</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.620(5)</td>
<td>2.648(4)</td>
<td>7.320(5)</td>
<td>1.476(4)</td>
</tr>
<tr>
<td>LDSVR</td>
<td>0.593(2)</td>
<td>2.393(2)</td>
<td>6.073(2)</td>
<td>1.342(2)</td>
</tr>
<tr>
<td>AA-ANN</td>
<td>0.593(2)</td>
<td>2.393(2)</td>
<td>6.073(2)</td>
<td>1.342(2)</td>
</tr>
</tbody>
</table>

Table 6: The experimental results on MORPH. Items in bold indicate the best performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Time (s)</th>
<th>Cost (s)</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC-LDL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.818(1)</td>
<td>6.777(1)</td>
<td>49.342(1)</td>
<td>0.716(2)</td>
</tr>
<tr>
<td>BCTable</td>
<td>0.885(4)</td>
<td>6.826(5)</td>
<td>49.925(4)</td>
<td>0.748(3)</td>
</tr>
<tr>
<td>BFGS-LDL</td>
<td>0.885(7)</td>
<td>6.812(4)</td>
<td>49.814(1)</td>
<td>0.897(5)</td>
</tr>
<tr>
<td>IIS-LDL</td>
<td>0.110(6)</td>
<td>0.892(6)</td>
<td>51.016(1)</td>
<td>1.119(6)</td>
</tr>
<tr>
<td>CPNN</td>
<td>0.108(3)</td>
<td>0.805(2)</td>
<td>49.529(2)</td>
<td>0.659(5)</td>
</tr>
<tr>
<td>LDSVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AA-ANN</td>
<td>0.076(1)</td>
<td>6.246(1)</td>
<td>45.101(1)</td>
<td>0.872(4)</td>
</tr>
</tbody>
</table>

Performance of these algorithms, respectively. Furthermore, the ranks of six measure values are also given in the parentheses right after the corresponding measure values, and the average value of these six ranks is denoted as Acc. Rank. Note that Avg. Rank of an algorithm is the average value between its Acc. Rank and Time Rank.

From Table 2-6, we can find that BC-LDL achieves the best accuracy performance on two datasets (i.e., s-BU_{3DFE} and Ren-CECPs), and achieves the second best performance on other datasets. Moreover, BC-LDL and BCTable can achieve better efficiency performance on the five evaluated datasets. Compared with other baselines, BC-LDL achieves the best comprehensive performance on the five datasets. The
comprehensive performance of BC-LDL on the five evaluated datasets demonstrates its effectiveness and efficiency.

The experimental results on s-BU_3DFE are reported in Table 2. We can see that our BC-LDL method outperforms other baseline methods, which demonstrates its superiority. Table 3 reports the experimental results on COPM. Here, we have a different observation: LDSVR achieves the better accuracy performance than BC-LDL. A reasonable explanation is that the application of the kernel trick makes LDSVR solve the LDL problem with the high-dimension features more effectively. Table 4 reports the experimental results on Twitter-LDL. We can find that AA-kNN achieves the best accuracy performance. The desirable results of AA-kNN can be attributed to the high-quality original features, which are extracted by VGGNet [Simonyan and Zisserman, 2014]. Table 5 reports the experimental results on Ren-CECps. Once again, BC-LDL yields better experimental results than other baseline algorithms. Furthermore, it is worth noting that the total time consumption of BCTable is least and its results are comparable to most LDL methods. The experimental results on MORPH are reported in Table 6. Although AA-kNN achieves the better accuracy performance, its time consumption of testing phase is much more than that of our methods.

To further demonstrate the efficiency of our algorithms, we also report the time ratio of the fastest compared method to our methods. The results are reported in Table 7. We can find that our algorithms are much faster than other baseline algorithms, especially on the large-scale datasets (i.e., Ren-CECps and MORPH).

### 3.3 Analysis on Binary Code Length

In this part, we study the evaluation results variation with respect to different code lengths on the five evaluated datasets. The code length is set to \{16, 32, \ldots, 256\} in actual experiments. Due to the space limitation, we only present the K-L and Intersection results. The results of other measures are similar. The evaluation results are reported in Figure 1-2. We have two observations. Firstly, longer binary codes can lead to better results on the medium-scale dataset (i.e., Twitter-LDL) and the large-scale datasets (i.e., Ren-CECps and MORPH). However, when the code length exceeds 128 bits, the performance of BC-LDL tends to converge. It proves that BC-LDL can achieve acceptable results with reasonably short binary codes, that is crucial in large-scale LDL. Secondly, as the code length increases, the performance improvement on s-BU_3DFE and COPM is not significant. When the code length exceeds 64 bits, the performance of BC-LDL even degrades to some extent. A possible reason for this phenomenon is that the size of these two datasets is small, and thus they cannot provide more useful information for longer binary codes.

### 4 Conclusion

In this paper, we have proposed a scalable LDL algorithm called BC-LDL, for dealing with the large-scale LDL problem. In BC-LDL, the label distribution information is integrated into the binary coding procedure for generating high-quality binary codes. Moreover, ANN search is conducted to find the training instances that have binary codes with a small Hamming distance from a testing instance, which can be efficiently computed via XOR operations. Then, the mean of the label distributions of all the neighboring instances is calculated as the predicted label distribution. Experimental results on five real-world datasets have demonstrated that BC-LDL is competitive with several state-of-the-art LDL methods.

### Acknowledgments

This research was supported by the National Key Research & Development Plan of China (No. 2017YFB1002801), the National Science Foundation of China (61622203), the Jiangsu Natural Science Funds for Distinguished Young Scholar
References


