Robust Textual Data Streams Mining Based on Continuous Transfer Learning

Bo Liu[*] Yanshan Xiao[†] Philip S. Yu[‡] Longbing Cao [§] Zhifeng Hao[¶]

Abstract

In textual data stream environment, concept drift can occur at any time, existing approaches partitioning streams into chunks can have problem if the chunk boundary does not coincide with the change point which is impossible to predict. Since concept drift can occur at any point of the streams, it will certainly occur within chunks, which is called random concept drift. The paper proposed an approach, which is called chunk level-based concept drift method (CLCD), that can overcome this chunking problem by continuously monitoring chunk characteristics to revise the classifier based on transfer learning in positive and unlabeled (PU) textual data stream environment. Our proposed approach works in three steps. In the first step, we propose *core vocabulary-based* criteria to justify and identify random concept drift. In the second step, we put forward the extension of LELC (PU learning by extracting likely positive and negative micro-clusters)[1], called soft-LELC, to extract representative examples from unlabeled data, and assign a confidence score to each extracted example. The assigned confidence score represents the degree of belongingness of an example towards its corresponding class. In the third step, we set up a transfer learning-based SVM to build an accurate classifier for the chunks where concept drift is identified in the first step. Extensive experiments have shown that CLCD can capture random concept drift, and outperforms state-of-the-art methods in positive and unlabeled textual data stream environments.

1 Introduction

Text classification plays an important role in summarizing textual information. Its main task is to set up a distinctive classifier to predict unseen examples. From the supervised learning perspective, both labeled positive and negative examples are required to build such a predictive classifier. In practice, it is not always possible to collect labeled negative examples [2]. For example, users may bookmark their favorite Web pages, but they are usually unwilling to mark boring pages. Another fact is that a positive class is usually more specific than a negative class. People can characterize their interests in detail, but they are often unable to specify what they do not like very well. Consequently, the problem of positive and unlabeled learning has attracted increasing attention [3]. In PU learning, only positively labeled training data and unlabeled data are available.

Depending on the nature of the representation models, the previous work on PU learning can be broadly classified into two categories: (1) static data-based methods [2, 4, 5, 6], in which a binary classifier and a one-class classifier are used to build a distinctive classifier for prediction; (2) the stream data-based method [1], in which LELC (PU learning by extracting likely positive and negative micro-clusters) extracts reliable negative examples, likely positive examples and likely negative examples from the unlabeled data; a classifier is thereafter built by combining positive and likely positive examples as positive class, and combining reliable negative examples and likely negative examples as negative class. Although this method has been found to be more accurate than most static data-based algorithms [1], the direct combination of examples ignores the difference between the reliable examples and the likely reliable examples, such as reliable negative examples and likely negative examples. This tends to limit the performance of LELC.

Despite much progress being made in LELC on mining textual data streams, most of the existing work holds an underlying assumption that concept drift occurs between chunks, and data in the same chunk share identical distribution. However, in real-world textual data streams, concept drift can occur at any point of the streams, it will certainly occur within chunks and that data in the same chunk may not share identical distribution, which is called random concept drift. This is because when a data stream system separates stream data into chunks, it is unlikely to know when and where the concept drift occurs. Consequently, existing approaches partitioning streams into chunks can have problems if the chunk boundary does not coincide with the change point which is impossible to predict. This type of issues, typically ignored in most previous work, is critically important and should be considered in the learning phase to build a more accurate classifier.

[*]Faculty of Automation, Guangdong University of Technology and Department of Computer Science, University of Illinois at Chicago. Email: csbliu@gmail.com

[†]Corresponding author. Faculty of Computer, Guangdong University of Technology. Email: xiaoyanshan@gmail.com

[‡]Department of Computer Science, University of Illinois at Chicago and Computer Science Department, King Abdulaziz University. Email: psyu@uic.edu

[§]Faculty of Engineering and IT, University of Technology, Sydney, Email: LongBing.Cao@uts.edu.au

[¶]Faculty of Computer, Guangdong University of Technology. Email: mazfhao@scut.edu.cn

This paper proposes a novel approach that can overcome this chunking problem by continuously monitoring chunk characteristics to revise the classifier based on transfer learning. Our proposed approach, which is called chunk level-based concept drift method (CLCD), can justify and identify possible random concept drift. The main contributions of our work are summarized as follows.

(1): We propose core vocabulary-based criteria to address random concept drift by analyzing the characteristic of the feature representation of the positive examples in textual data streams. Our method can automatically justify possible random concept drift and vaguely identify where it occurs if it exists.

(2): We introduce soft-LELC, on top of the LELC method [1], by assigning a confidence score for each extracted representative example from unlabeled data. The confidence score denotes the degree of belongingness of an example towards its corresponding class. In this way, examples contribute differently to the classifier construction based on their confidence scores, such that we can build a more accurate classifier for prediction.

(3): We extend transfer learning-based SVM to build a classifier for the chunks where random concept drift is identified in the first step of Algorithm 1. In this phase, the extracted examples, as well as their confidence scores, are incorporated into the learning phase. In case that no random concept drift is found in the first step of Algorithm 1, we extend the standard SVM to incorporate the extracted examples and their confidence scores into the learning.

(4): Finally, we conduct experiments on the textual data streams generated from real-world textual datasets to evaluate the performance of our proposed CLCD approach. The results show that CLCD can capture random concept drift and can outperform state-of-the-art PU learning methods.

The advantage of our proposed approach can be summarized as follows. (1): Our proposed approach can vaguely detect and handle concept drift occurs within a data chunk and construct a more accurate classifier in comparison to the existing data stream methods. (2): Compared with existing methods for positive and unlabel learning, the soft-LELC, on top of LELC, can be efficiently converted into transfer-learning-based classifier by incorporated the different distribution data and the confidence scores into learning and delivers accurate classifier at the third step of of Algorithm 1.

For clarity, the basic notations and their meanings are presented in Table 1.

Table 1: Basis notations and meaning

Notions	Meaning
$\mathbf{x}_i$	the i^{th} training example
f_k	the k^{th} feature of source data
V	features set of source data, $f_k \in V$
$\|S\|$, r	sample size of S and concept drift rate of stream
	for current chunk
D_c	stream chunk between t_{t-1} and t_t
PD_c	labeled positive samples set
UD_c	unlabeled examples set
	for j^{th} portion
l	number of portions for current chunk
D_c^j	the j^{th} portion of current chunk
PD_c^j	the positive examples set in D_c^j
UD_c^j	the unlabeled examples set in D_c^j, $D_c^j = PD_c^j \bigcup UD_c^j$
VP_c^j	positive features set of D_c^j
$H_c^j(f_k)$	feature strength of feature f_k
θ_c^j	average feature strength where $f_k \in V$
	for neighboring portions
$H_c^{j-j+1}(f_k)$	feature strength of f_k between PD_c^j and PD_c^{j+1}, $f_k \in VP_c^j \bigcup VP_c^{j+1}$
Θ_c^j	similarity of PV_j and PV_{j+1}
	for first category data [1]
D_c^I	The set of the first category data
PD_c^I	positive documents set of D_c^I
UD_c^I	unlabeled documents set of D_c^I
RU_c^I	negative documents set of D_c^I
LU_c^I	$LU_c^I = UD_c^I - RU_c^I$

2 Related Work

We review the previous PU learning methods and data streams as follows.

2.1 PU Learning for Text Mining

The previous work on PU learning can be broadly classified into two categories: (1) static data-based methods, and (2) the stream data-based method.

In the static data-based methods, a binary classifier and a one-class classifier can be used to predict unseen examples. In the binary classifier-based methods [3, 4, 7, 8], a set of representative examples is extracted from the unlabeled data in step one, and a classifier is then built in step two. Xiao [9] introduces similarity weights to each sample in the unlabeled data and constructs an classifier to improve the performance of positive and unlabeled learning; on the other hand, it does not handle the concept drift in data streams. .

For the stream data-based method [1], LELC [1] extracts reliable negative examples, likely positive examples and likely negative examples from unlabeled data in step one, and then builds an SVM classifier in step two. The positive and likely positive examples are considered as positive class, and negative and likely negative examples are treated as negative class. Although LELC has been shown to be more accurate than other methods, this direct combination ignores the difference between positive examples and likely positive examples, and the difference between negative examples and likely negative examples. However, the reliable examples (positive and reliable negative examples) and likely examples (likely positive and likely negative examples) should contribute differently to the construction of the SVM-based classifier. This always limits the performance of the LELC method.

2.2 Data Stream Recently, advanced data stream technologies have been used to process large amounts of high frequency data such as textual news streams [10]. Due to its crucial position in industry applications, data stream mining has been investigated [10, 11]. We briefly introduce the previous data stream work according to the assumption on concept drift.

In the first category [1, 12], they typically hold the assumption that concept drift occurs between chunks, and that data in the same chunk share identical distribution. This is because, for most data stream problems, it is difficult to know whether and where concept drift may occur [13]. Consequently, a classifier is built for each chunk by considering the chunk data as a whole. The advantage of these methods is that, they can address concept drift occurring between chunks very well. However, they may have problem if the chunk boundary does not coincide with the change point which is impossible to predict.

In another category, the work in [14] assumes that concept drift occurs from the last 10% examples at the tail of each chunk. However, this assumption of a fixed drift point for each chunk is not always realistic. This is because concept drift may occur anywhere within a chunk in addition to occurring at the end of the chunk. Another reason is that, for some chunks, there may not exist concept drift in them. For the above reasons, it is necessary to explore new techniques to justify and identify random concept drift in the data stream environment.

This paper proposes *core vocabulary-based* criteria to automatically justify and identify random concept drift and then vaguely identifies where the concept drift occurs if there exists.

3 Preliminary

3.1 Problem Definition Suppose we have a series of textual data stream chunks $D_1, D_2, \ldots, D_c$, where each chunk $D_t (t = 1, 2, \ldots, c)$ contains the data that arrived between time period of t_{t-1} and t_t. Here, D_c is called the current chunk and the yet-to-come data chunk (denoted as D_{c+1}) is called target chunk. Due to the fact that data in the same chunk may not share identical distribution, this makes textual data stream learning far more difficult than previous data stream learning.

To describe this scenario, a conceptual view is shown in Figure 1. For the current chunk, we broadly category data into two categories. As illustrated in Figure 1, data in the first category (type 1) has a similar distribution to the target chunk data, and examples in the second category (type 2) share the same data distribution as the target chunk data. The assumption is the same as the previous work [14]. Considering the temporal correlations of the data streams

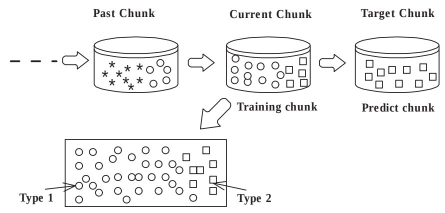


Figure 1: A conceptual view of the example type categorization for current chunk

[10, 14], the second category data might be close to the target chunk. Since data distribution of type 1 is similar to that of type 2, the work in [14] assumes concept drift occurs from the last 10% examples at the tail of each chunk and constructs transfer learning classifier on both types of data. Our proposed approach automatically detects the concept drift and builds transfer learning classifier to build a more accurate classifier.

For this problem setting, we assume the number of labeled positive examples depends on the labeling speed the experts offer. Assume the data streams flow at a speed of s_d examples per second, the coming data is buffered and labeled by experts with a speed of s_l examples per second.

One key challenge in positive and unlabeled textual data streams is how to justify and identify random concept drift. We will introduce our proposed core vocabulary-based criteria to address this in Section 4.1.

3.2 Confidence Score Model For a given instance $\mathbf{x}$, the confidence score model is defined as $\{\mathbf{x}, y, m(\mathbf{x})\}$, where y denotes the class label. The confidence score $m(\mathbf{x})$ represents the degree of belongingness of instance $\mathbf{x}$ towards class y. In general, $0 \leq m(\mathbf{x}) \leq 1$ holds true and $m(\mathbf{x}) = 1$ means example $\mathbf{x}$ completely belongs to class y. By assigning a confidence score to each example, we can ensure that the examples make different contributions to the construction of the classifier.

4 Proposed Approach

In this Section, we introduce our proposed approach for positive and unlabeled textual data streams. Due to the fact that concept drift may occur within chunks and data in the same chunk may not share identical distribution, as illustrated in Figure 1, this creates a new challenge for data stream problems. Our proposed CLCD approach consists of four steps, as illustrated in Algorithm 1. In the following, we exhibit the four steps in detail. For simplicity, we detail each step for the current chunk D_c. We can generalize each step on other chunks.

4.1 Core Vocabulary-based Criteria for Identifying Two Categories of Data The determination of the two categories of data depends on the concept drift rate [14]. Assuming that data come uniformly between t_{c-1} and t_c, the number of the second category of data has a reverse proportion to r with a coefficient γ. In theory, around $\gamma \cdot r^{-1}|D_c|$ examples

[1]Similar notions are defined for the second category data, i. e., $D_c^{II}, PD_c^{II}, UD_c^{II}, RU_c^{II}, LU_c^{II}$.

Algorithm 1 Outline of CLCD Approach

```
1: Step 1: Propose core vocabulary-based criteria to justify whether
   concept drift exists in the current chunk. If there exists, vaguely identify
   the concept drift point;
2: Step 2: Put forward soft-LELC to introduce a confidence score for each
   extracted example on top of the LELC method [1];
3: if there exists random concept drift then
4:     For the two types of data distributions, use soft-LELC on each data
       distribution to extract representative examples and assign confidence
       scores for them;
5: else
6:     Use soft-LELC on the whole chunk data to extract representative
       examples and assign confidence scores for them;
7: end if
8: Step 3:
9: if there exists random concept drift then
10:    Build a new transfer learning-based SVM to incorporate the two
       types of data distributions into learning;
11: else
12:    Build an SVM-based classifier on the whole chunk by incorporating
       the extracted examples and their confidence scores into a learning
       phase;
13: end if
14: Step 4: Integrate the classifiers from the current and historical chunks
    to form an ensemble classifier for prediction.
```

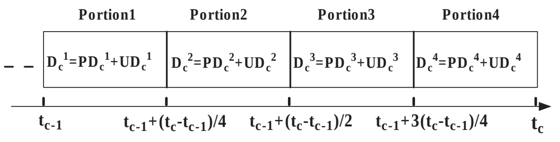


Figure 2: Example of dividing the current chunk into four portions referring to the time interval.

belong to the second category of data while $(1-\gamma\cdot r^{-1})|D_c|$ examples belong to the first category of data. However, we may have no knowledge about r and γ in practice, thus this method can not be used directly. As an alternative, we propose *core vocabulary-based criteria* to justify whether concept drift exists within the current chunk and, if it does, to vaguely identify where it occurs by considering the characteristic of features representation of the positive examples.

4.1.1 Portions of the Current Chunk First, for the current chunk D_c, we divide data into l portions by referring to the time interval between t_{c-1} and t_c so that each portion contains both positive and unlabeled examples. More specifically, let $\Delta_c = (t_c - t_{c-1})/l$ and assume the data coming between $t_{c-1} + (j-1)\cdot\Delta_c$ and $t_{c-1} + j\cdot\Delta_c$ are stored in D_c^j $(j = 1, 2, \ldots, l)$. Consequently, D_c^j consists of two sets: PD_c^j which contains labeled positive examples, and UD_c^j which consists of unlabeled examples. As illustrated in Figure 2, assume we divide the current chunk data into four portions; we then have four sets D_c^j $(j = 1, 2, 3, 4)$.

4.1.2 Positive Features Extraction For each portion D_c^j, we identify the *positive features* which frequently appear in PD_c^j according to the notion of core vocabulary [2]. It is generally believed that an example belonging to positive examples (D_c^j) must possess some features contained in the *core vocabulary* of D_c^j [2], and the set of *positive features* for D_c^j is denoted by VP_c^j. Following the operation in [2], we extract the *positive features* which appear frequently in positive examples (D_c^j) while occur less frequently in unlabeled examples (UD_c^j) as follows.

For each portion D_c^j, we calculate the *feature strength* of a particular feature f_k by measuring the differences of the normalized examples frequency between PD_c^j and UD_c^j

$$(4.1)\quad H_c^j(f_k) = \frac{n_{PD_c^j}(f_k) - min_{PD_c^j}}{max_{PD_c^j} - min_{PD_c^j}} - \frac{n_{UD_c^j}(f_k) - min_{UD_c^j}}{max_{UD_c^j} - min_{UD_c^j}},$$

where $n_{PD_c^j}(f_k)$ and $n_{UD_c^j}(f_k)$ represent the number of examples that contain f_k in PD_c^j and UD_c^j respectively, $max_{PD_c^j}$ and $min_{PD_c^j}$ are maximum and minimum values of $n_{PD_c^j}(f_k)$ for $f_k \in PD_c^j$. A similar formula applies to $max_{UD_c^j}$ and $min_{UD_c^j}$.

After that, the *positive features* are identified by extracting f_k such that $H_c^j(f_k) > \theta_c^j$, where

$$(4.2)\qquad \theta_c^j = \frac{1}{N_{PD_c^j}} \sum_{f_k \in PD_c^j} H_c^j(f_k)$$

when $N_{PD_c^j}$ is the number of features in PD_c^j.

By utilizing the robust positive features extraction method in [2], we successfully extract positive features for portion D_c^j and put them into VP_c^j.

4.1.3 Boundary Determination for Two Categories of Data We compare the similarity of the *positive features* between two neighboring portions to identify the vague boundary of the two categories of data in the current chunk. More specifically, for the two neighboring portions PD_c^j and PD_c^{j+1}, the *feature strength* of feature f_k $(f_k \in VP_c^j \cup VP_c^{j+1})$ between PD_c^j and PD_c^{j+1}, denoted as $H_c^{j-j+1}(f_k)$, is calculated in the same way as Equation (4.1). According to Equation (4.2), the similarity of the *positive* features between two neighboring portions is defined as follows.

(4.3)
$$\Theta_c^j = 1 - \frac{1}{|VP_c^j \cup VP_c^{j+1}|} \sum_{f_k \in VP_c^j \bigcup VP_c^{j+1}} |H_c^{j-j+1}(f_k)|.$$

The possible vague boundary of two categories of data is between portion j and $j+1$ where j equals

$$(4.4)\qquad n = \underset{j}{arg\,max}(\Theta_c^j)\ \ j = 1, 2, \ldots, l-1.$$

Remark: Equation (4.4) is robust since it captures the idea that the similarity of two sets of positive features which are from the same core vocabulary distribution is always higher than that of two sets from different core vocabulary distributions. For example, as illustrated in Figure 2, assume the user's interest is "computer" at the first two portions and it turns into "recreation" at the third and fourth portions.

When we calculate the feature strength between the first two portions, $H_c^{1-2}(f_k)$ should ideally be zero because the positive features of two portions come from the same core vocabulary distribution; however, it might not equal zero due to the fact that the positive features of each portion might contain part of the computer information. For the same reason, $H_c^{3-4}(f_k)$ might not equal zero. However, since the user's interest changes between the second and third portion, the positive features of the second portion should share little with those of the third portion. This is because the core vocabulary distributions of "computer" and "recreation" are quite different. According to the characteristic of Equation (4.1) discussed in footnote 2, $H_c^{2-3}(f_k)$ is either strictly towards to PD_c^2 or towards PD_c^3. Therefore, Θ_c^2 will be a comparably larger value than Θ_c^1 and Θ_c^3.

From (4.4), we can obtain the vague boundary of two types of data if the user's interest occurs within chunks. In the case that there is no random concept drift and we still return n, we use the following criteria to judge whether concept drift exists within the current chunk. We first calculate the average of the other Θ_c^j except for Θ_c^n

$$\overline{\Theta}_c = \frac{1}{l-2} \sum_{j \neq n} \Theta_c^j. \qquad (4.5)$$

We then calculate the ratio of $\overline{\Theta}_c$ and Θ_c^n, and if the ratio is less than a pre-specific tolerance score λ, we consider that the user's interest drifts within the chunk. That is, (4.6) holds true

$$\overline{\Theta}_c / \Theta_c^n < \lambda. \qquad (4.6)$$

Otherwise, it is considered that no concept drift exists within the chunk. The tolerance score λ reflects the tolerance on the concept drift, and λ is a value between $(0, 1)$. The smaller λ is, the larger tolerance we have on the concept drift within the chunk; the larger λ is, the smaller tolerance we take on the concept drift.

The pseudo code is presented in Algorithm 2. If the returned variable *bool* equals 1, it is considered that concept drift exists within the current chunk. We then let $D_c^I = D_c^1 \cup \cdots \cup D_c^n$ and $D_c^{II} = D_c^{n+1} \cup \cdots \cup D_c^l$ such that $PD_c^I = PD_c^1 \cup \cdots \cup PD_c^n$, $UD_c^I = UD_c^1 \cup \cdots \cup UD_c^n$, $PD_c^{II} = PD_c^{n+1} \cup \cdots \cup PD_c^l$ and $UD_c^{II} = UD_c^{n+1} \cup \cdots \cup UD_c^l$. In this case, D_c^I and D_c^{II} contain two types of data distributions respectively. If *bool* returns 0, we consider that concept drift does not exist within the current chunk.

Algorithm 2 Justification and identification of two types of data.

```
Input: $D_c$, $l$, $\lambda$; // current chunk, portion number and tolerance score
Output: bool.// bool=1 means concept drift exists within chunks.
 1: Portion $D_c$ into $l$ portions referring to time interval;
 2: Obtain $D_c^j$, $PD_c^j$ and $UD_c^j$ $(j = 1, \ldots, l)$.
 3: for (j=1; j++; j≤ l) do
 4:     Calculate the feature strength of feature $f_k$ between $PD_c^j$ and $UD_c^j$
        according to Equation (4.1);
 5:     Compute the average feature strength by Equation (4.2);
 6:     Identify positive features and put them into set $VP_c^j$;
 7: end for
 8: for (j=1; j++; j≤ l-1) do
 9:     Calculate the similarity of positive features between $PD_c^j$ and
        $PD_c^{j+1}$ by Equation (4.3);
10:     Return the possible vague boundary of two types of data by (4.4);
11: end for
12: Obtain the average of $\Theta_c^j$ except for $\Theta_c^n$ by Equation (4.5).
13: if Equation (4.6) holds true then
14:     bool=1;
15: else
16:     bool=0;
17: end if
18: Return bool.
```

4.2 Representative Example Extraction and Confidence Score Assignment

We put forward the soft-LELC method, on top of the LELC method, to extract representative examples from the unlabeled data and assign a confidence score for each example. If random concept drift is found, that is *bool* returns 1 in Algorithm 2, we perform soft-LELC on D_c^I and D_c^{II} respectively. If no concept drift exists within the current chunk, that is *bool* returns 0 in Algorithm 2, we perform soft-LELC on the whole chunk data D_c. For simplicity, we introduce soft-LELC method on D_c^I as follows.

The soft-LELC method extracts the reliable negative examples, likely positive and likely negative examples from the unlabeled examples the same as the LELC method [1]. Unlike LELC, each example $\mathbf{x}_i$ is assigned a confidence score which represents the degree of belongingness towards its corresponding class. The example with its confidence score is denoted by $(\mathbf{x}, y, m(\mathbf{x}))$. Since the existing positive examples belong completely to the positive class, each positive example with its confidence score is denoted as $(\mathbf{x}_i, +1, 1)$.

4.2.1 Reliable Negative Example Extraction

As with LELC, we integrate both the Spy extraction [3] and the Rocchio extraction [7] to extract the most reliable negative examples. Examples are classified as reliable negative examples only if both methods consider them as negative examples at the same time.

We put the extracted reliable negative examples from UD_c^I into set RU_c^I. We then assign confidence score 1 to each extracted reliable example, because we can be confident that each example belongs completely to the negative class. Thus, each example with its confidence score in RU_c^I is denoted as $(\mathbf{x}_i, -1, 1)$. After this, we let $LU_c^I = UD_c^I - RU_c^I$ and LU_c^I consists of the remaining unlabeled examples.

4.2.2 Remaining Unlabeled Examples Assignment

Although positive examples and the extracted negative examples can directly build a classifier for prediction. However, it is worth extracting likely positive and likely negative examples from LU_c^I and take them into the learning phase to build an accurate classifier. The pseudo code for the extraction of the likely positive and negative examples and confidence score assignment is presented in Algorithm 3.

Algorithm 3 Extraction of the likely positive and negative examples and confidence score assignment.

```
Input: $RU_c^I, PD_c^I, LU_c^I$ // reliable negative examples set, reliable positive examples set and rest unlabeled examples
Output: $(\mathbf{x}_i, y_i, m(\mathbf{x}_i)), \mathbf{x}_i \in LU_c^I$ //
1: Cluster the reliable negative examples in $RU_c^I$ to $m$ micro-clusters: $RU_{c1}^I, RU_{c2}^I, \ldots, RU_{cm}^I$. The number of $m$ is set as $m = |RU_c^I|/(|RU_c^I| + |LU_c^I|) * k$;
2: for (i=1; i++; i≤m) do
3:     $\overrightarrow{\mathbf{p}_i} = \alpha \frac{1}{|PD_c^I|} \sum_{\mathbf{x} \in PD_c^I} \frac{\mathbf{x}}{\|\mathbf{x}\|} - \beta \frac{1}{|RU_{ci}^I|} \sum_{\mathbf{x} \in RU_{ci}^I} \frac{\mathbf{x}}{\|\mathbf{x}\|}$;
4:     $\overrightarrow{\mathbf{n}_i} = \alpha \frac{1}{|RU_{ci}^I|} \sum_{\mathbf{x} \in RU_{ci}^I} \frac{\mathbf{x}}{\|\mathbf{x}\|} - \beta \frac{1}{PD_c^I} \sum_{\mathbf{x} \in PD_c^I} \frac{\mathbf{x}}{\|\mathbf{x}\|}$;
5: end for
6: Cluster the remaining unlabeled examples in $LD_c^I$ to $n$ micro-clusters: $LD_{c1}^I, LD_{c2}^I, \ldots, LD_{cn}^I$. The number of $n$ is set as $n = |LU_c^I|/(|RU_c^I| + |LU_c^I|) * k$;
7: for each micro-cluster $LU_{ci}^I$ do
8:     pvote=0; nvote=0;
9:     for each example $\mathbf{x} \in LD_{ci}^I$ do
10:        if $max_{j=1}^m Sim(\mathbf{x}, \overrightarrow{\mathbf{p}_j}) > max_{j=1}^m Sim(\mathbf{x}, \overrightarrow{\mathbf{n}_j})$ then
11:            pvote=pvote+1;
12:        else
13:            nvote=nvote+1;
14:        end if
15:    end for
16:    if $pvote > nvote$ then
17:        for each example $\mathbf{x}_i$ in $LU_{ci}^I$ do
18:            The label of $\mathbf{x}_i$ is set positive: $y_i = 1$;
19:            $m(\mathbf{x}_i) = pvote/(pvote + nvote)$;
20:        end for
21:    else
22:        The label of $\mathbf{x}_i$ is set negative: $y_i = -1$;
23:        $m(\mathbf{x}_i) = nvote/(pvote + nvote)$.
24:    end if
25: end for
26: Output: $(\mathbf{x}_i, y_i, m(\mathbf{x}_i))$
```

In Algorithm 3, we use the same technique as LELC to separate the unlabeled examples from steps 1 to 6. Step 1 clusters the examples in RU_c^I into micro-clusters. In the k-means method, cosine similarity [15] is used for similarity calculation between $\mathbf{x}_i$ and $\mathbf{x}_j$, that is $Sim(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \cdot \|\mathbf{x}_j\|}$. Steps 2 to 4 construct positive and negative representative prototypes p_i and n_i $(i = 1, 2, \ldots, l)$, and $\alpha = 16$ and $\beta = 4$ are recommended in [16]. Step 6 clusters the unlabeled examples into micro-clusters where the examples in the same micro-cluster are self-similar. In the clustering steps 1 and 6, k is set 30 as recommended in [1].

Soft-LELC and LELC have different assignment strategies for the examples in the micro-cluster. LELC considers the examples in the same micro-cluster to be self-similar, and assigns these examples to a class, in which most of the examples are similar. For example, assume that l_t examples exist in a micro-cluster, in which l_p examples are similar to the closest positive prototype $\overrightarrow{\mathbf{p}_i}$, and $l_t - l_p$ examples are similar to the closest negative prototype $\overrightarrow{\mathbf{n}_i}$. If $l_p > l_t - l_p$ holds, then l_t examples are assigned to positive class; If $l_p < l_t - l_p$ holds, they belong to negative class. Soft-LELC, however, assigns the examples into the class with a certain confidence score. For example, if $l_p > l_t - l_p$, the l_t examples are assigned into positive class with a confidence score l_p/l_t. Otherwise, the examples are classified into negative class with a confidence score $(l_t - l_p)/l_t$.

Steps 7 to 25 show the confidence score assignment for the extracted examples in detail. So far, we have assigned each example in LU_c^I into positive class or negative class with a confidence score. The next step is to build a predictive classifier using the extracted representative examples and their confidence scores.

4.3 Classifier Construction Since each chunk may or may not have concept drift, we put forward transfer learning-based SVM for the chunks where concept drift occurs, and extend the standard SVM for the chunks in which no concept drift exists.

4.3.1 Transfer Learning-based Classifier If random concept drift is found, that is *bool* returns 1 in Algorithm 2, the soft-LELC method is performed on D_c^I and D_c^{II} respectively. In general, we need to use the classifier built on D_c^{II} for prediction since the data in D_c^{II} are more likely to share similar distribution as the target chunk data. In practice, the samples in D_c^{II} may be limited compared with those in D_c^I, which are insufficient to construct an accurate classifier. In this case, we propose the extension of the transfer learning-based SVM [17] to refine the classifier boundary by transferring knowledge from D_c^I to D_c^{II}. Meanwhile, we incorporate the extracted examples and their confidence scores into the learning phase.

Suppose we have two tasks, that is, to train SVM on D_c^I for task one and on D_c^{II} for task two:

$$w_1 = w_o + v_1 \quad and \quad w_2 = w_o + v_2, \tag{4.7}$$

where w_1 and w_2 are parameters of the SVM for task one and task two, respectively. w_o is a common parameter while v_1 and v_2 are specific parameters. By assuming $f_1 = w_1 \cdot \mathbf{x}$ and $f_2 = w_2 \cdot \mathbf{x}$ to be two hyperplanes for D_c^I and D_c^{II}. w_1 and w_2 can be denoted as $w_t = w_0 + v_t, t = 1, 2$ and the extended version of transfer learning-based SVM can be written as follows.

$$\begin{aligned} & min \;\; J(w_o, v_t, \xi) \\ & = \tfrac{1}{2}\|w_o\|^2 + \textstyle\sum_{t=1}^2 C_t \|v_t\|^2 + C(\sum m(\mathbf{x}_{1i})\xi_{1i} \\ & \qquad + \textstyle\sum m(\mathbf{x}_{2j})\xi_{2j}) \\ & s.t. \;\; y_{1i}((w_o + v_1) \cdot \mathbf{x}_{1i}) \geq 1 - \xi_{1i}, \quad \mathbf{x}_{1i} \in D_c^I \\ & \quad\;\; y_{2j}((w_o + v_2) \cdot \mathbf{x}_{2j}) \geq 1 - \xi_{2j}, \quad \mathbf{x}_{2j} \in D_c^{II} \\ & \qquad\qquad \xi_{1i} \geq 0 \quad \xi_{2j} \geq 0, \end{aligned} \tag{4.8}$$

where parameters C_1 and C_2 control the preference of the two tasks. If $C_1 > C_2$, task 1 is preferred to task 2; otherwise, task 2 is preferred to task 1. Parameters ξ_{1i} and

ξ_{2j} are defined as measure of error, the terms $m(\mathbf{x}_{1i})\xi_{1i}$ and $m(\mathbf{x}_{2j})\xi_{2j}$ can be therefore considered as measure of error with different weighting factors. Note that a smaller value of $m(\mathbf{x}_{1i})$ could reduce the effect of the parameter ξ_{1i} in Equation (4.8), such that the corresponding data example ξ_{1i} becomes less significant in the training.

As discussed in [17], Lagrangian function can be used to resolve Equation (4.8). We then obtain two hyperplanes for D_c^I and D_c^{II} respectively, i.e., $f_1(\mathbf{x}) = w_1 \cdot \mathbf{x} + b_1$ and $f_1(\mathbf{x}) = w_1 \cdot \mathbf{x} + b_2$. Since the data in D_c^{II} are likely to share similar distribution as the data in the target chunk, we use the refined classifier built on the D_c^{II} for prediction, and denote it as l_c.

4.3.2 Extended SVM Construction If no concept drift is found in the current chunk, that is *bool* returns 0 in Algorithm 2, the standard SVM is extended to take the extracted examples and their confidence scores into the learning:

$$\begin{aligned} &min\ \ J(w_o, \xi) = \tfrac{1}{2}\|w_o\|^2 + C\textstyle\sum m(\mathbf{x}_i)\xi_i \\ &\quad s.t.\ \ y_i(w_o \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \\ &\qquad\quad \xi_i \geq 0 \quad \mathbf{x}_i \in D_c, \end{aligned} \tag{4.9}$$

where $w_o \cdot \mathbf{x}_i + b$ is a hyperplane. By using Lagrangian function, we can obtain the SVM classifier.

Since no concept drift is found in the current chunk, we use the obtained classifier for prediction and denote it as l_c.

4.3.3 Computation Complexity Analysis Suppose the computation complexity of the standard SVM is in proportion to $O(p)^T$ where $T \leq 2$ and p is the sample size for binary SVM [18]. Since problem (4.8) requires solving a standard SVM problem with $2m$ training data [17], where $m = max\ (|D_c^I|, |D_c^{II}|)$, the computation complexity is $O(2m)^T$. Additionally, the complexity of solving problem (4.9) is the same as SVM, i.e., $O(|D_t|)^T$.

4.4 Ensemble Classifier Construction After the classifier l_c is trained from the current chunk D_c, this classifier and the most recent $k-1$ classifiers $(l_{c-k+1}, l_{c-k+2}, \ldots, l_{c-1})$, which are built on the historical chunks, form an ensemble classifier l_E. By referring to the method in [10], the weight value of each classifier is calculated by computing its classification error on the data in D_c^{II} (if random concept drift is determined in Algorithm 2) or on the data in D_c (if no random concept drift is found in Algorithm 2). After the ensemble classifier l_E has been formed, it is used to predict the data in the target chunk.

5 Experiments

5.1 Baselines and Metric We evaluate our approach CLCD method empirically. For comparison, another three textual data stream methods are used as baselines. (1) The first method is LELC [1] which is state-of-the-art PU learning algorithms for handling concept drift and has been found to be more accurate than traditional PU learning methods [1]. (2) The second baseline is the degenerated version of our approach, called soft-LELC, which starts from the second step of CLCD without justifying and identifying concept drift; that is, it assumes concept drift occurs between chunks. This baseline is used to show the effectiveness of confidence score assignment by comparing the performance of LELC and soft-LELC methods. (3) The third baseline is derived from [14], which assumes concept drift occurs from the last 10% examples at the tail of each chunk. We use soft-LELC and transfer learning-based SVM to build a classifier for each chunk, similar to CLCD. This method is called TSVM. This baseline is used to show the improvement of CLCD over the method assuming a fixed concept drift for each chunk.

The performance of text classification is typically evaluated in terms of F-measure [3] which trades off precision p and recall r: $F = 2pr/(r+p)$. We use F-measure as metric in the experiments.

5.2 Dataset Description In the previous data streams study [1, 19], they always generate data streams from real-world UCI datasets. The same as them, we use two typical real-world textual datasets, which have been previously used by other researchers for textual data streams [12, 1], in the experiments:

(1): Newsgroup-20[2]: This dataset contains 20 different sub-categories of news; each sub-category contains about 1000 news items, these 20 sub-categories belong to five seven categories: “alt”, “comp”, “misc”, “rec”, “sci”, “soc” and “talk”. We ignore the news having multi labels in the experiments. (2): Web-KB[3]: There are 8,282 web pages and seven categories, i.e., “student”, “faculty”, “staff”, “department”, “course”, “project” and “other”. Each Web page belongs to one category. Web-KB is slightly skewed, “other” category has more web pages compared with others. We use the first six categories in the experiments.

Each document is represented by TFIDF features. By referring to the operation in [12, 1], we generate the data streams from Newsgroup-20 and Web-KB datasets and concept drifts occur in them. For the data streams, we only label a portion of the positive examples, this operation refers to the labeled and unlabeled data stream learning [12]. For the unlabeled positive examples and negative examples are considered as unlabeled data.

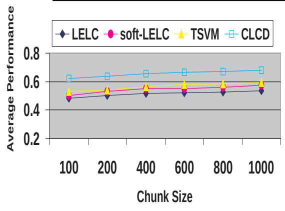
5.3 Experiment Setting The linear kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ is used since it always performs excellently for text classification tasks [20]. All the experiments are conducted on a laptop with a 2.8 GHz processor and 3GB DRAM. We implement CLCD using the VC++ 6.0 developing environment and use LibSVM to solve the standard SVM for LELC and the extended SVM.

In the LELC, we use the recommended parameter values in [1], introduced in Section 4.2. Soft-LELC adopts the same

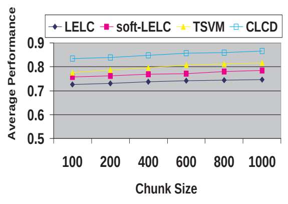
[2]Available at http://kdd.ics.uci.edu/databases/20newsgroups
[3]Available at http://www.cs.cmu.edu/ webkb/

Table 2: Average performance and standard deviation of the fifty chunks with respect to different concept drift rates.

Data Set	Method	Concept Drift					
		r=20%	r=30%	r=40 %	r=50%	r=60%	r=70%
Newsgroup-20	LELC	$0.547_{\pm 0.067}$	$0.517_{\pm 0.062}$	$0.478_{\pm 0.065}$	$0.445_{\pm 0.085}$	$0.423_{\pm 0.102}$	$0.397_{\pm 0.123}$
	Soft-LELC	$0.568_{\pm 0.068}$	$0.549_{\pm 0.058}$	$0.517_{\pm 0.06}$	$0.494_{\pm 0.074}$	$0.475_{\pm 0.098}$	$0.429_{\pm 0.107}$
	TSVM	$0.598_{\pm 0.061}$	$0.564_{\pm 0.056}$	$0.559_{\pm 0.063}$	$0.543_{\pm 0.073}$	$0.524_{\pm 0.103}$	$0.478_{\pm 0.112}$
	CLCD	$\mathbf{0.692}_{\pm 0.057}$	$\mathbf{0.654}_{\pm 0.052}$	$\mathbf{0.649}_{\pm 0.055}$	$\mathbf{0.638}_{\pm 0.069}$	$\mathbf{0.62}_{\pm 0.967}$	$\mathbf{0.606}_{\pm 0.096}$
Web-KB	LELC	$0.753_{\pm 0.057}$	$0.737_{\pm 0.065}$	$0.722_{\pm 0.071}$	$0.683_{\pm 0.072}$	$0.67_{\pm 0.101}$	$0.609_{\pm 0.127}$
	Soft-LELC	$0.78_{\pm 0.053}$	$0.768_{\pm 0.062}$	$0.758_{\pm 0.069}$	$0.725_{\pm 0.068}$	$0.708_{\pm 0.097}$	$0.638_{\pm 0.108}$
	TSVM	$0.812_{\pm 0.055}$	$0.797_{\pm 0.065}$	$0.783_{\pm 0.067}$	$0.752_{\pm 0.062}$	$0.73_{\pm 0.085}$	$0.691_{\pm 0.094}$
	CLCD	$\mathbf{0.867}_{\pm 0.051}$	$\mathbf{0.848}_{\pm 0.059}$	$\mathbf{0.841}_{\pm 0.062}$	$\mathbf{0.827}_{\pm 0.065}$	$\mathbf{0.822}_{\pm 0.075}$	$\mathbf{0.805}_{\pm 0.089}$

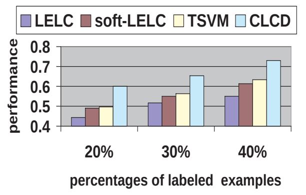


(a) Newsgroup-20

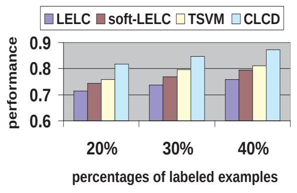


(b) Web-KB

Figure 3: Performance comparison with different chunk size.



(a) Newsgroup-20



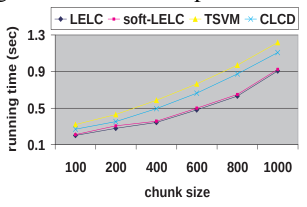
(b) Web-KB

Figure 4: Performance comparison with different percentages of labeled examples.

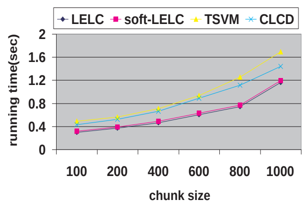
parameter settings as LELC. LELC uses the standard SVM and soft-LELC uses the extended SVM (Equation (4.9)), we allow parameter C in SVM classifier to be from 1 to 1000. For the CLCD, λ is set as 0.5. The number of the portions l is chosen as 10, and we will investigate the performance sensitivity to l in the experiment. In the transfer learning-based model (Equation (4.8)), C_1 and C_2 control the tradeoff between the global optimal boundary w_o and the local optimal boundary. If we assign a relatively large value to C_2, the global optimal solution will bias towards task 2. Because the distribution of data in D_c^{II} is likely to be more similar to the target chunk data compared with the data in D_c^I, we let $C_2/C_1 = 10 * (l - n)/n$, where n is the returned concept drift place in Algorithm 2. The C in transfer learning-based SVM (Equation (4.8)) is also chosen from 1 to 1000. In the ensemble framework, the number of ensemble classifiers is set $k = 10$.

5.4 Performance Comparison We compare LELC, soft-LELC, TSVM and CLCD with respect to different concept rates. Table 2 reports the average performance and standard deviation of the fifty chunks (from the tenth chunk to the sixtieth chunk) with respect to different concept drift rates from $r = 20\%$ to $r = 70\%$. Here about 30% of positive examples are labeled and chunk size is fixed at 400. It is observed that soft-LELC outperforms LELC at all times. This is because soft-LELC, built on top of LELC, incorporates the confidence score of each extracted example to build a more accurate classifier. Meanwhile, we find that CLCD performs significantly better than others. This indicates the effectiveness of the core vocabulary-based criteria to justify and identify random concept drift.

We also observe that the performance of the four methods decreases as concept drift rate r becomes larger. In general, larger r will lead to more concept drifts in the data



(a) Newsgroup-20



(b) Web-KB

Figure 5: Running time comparison

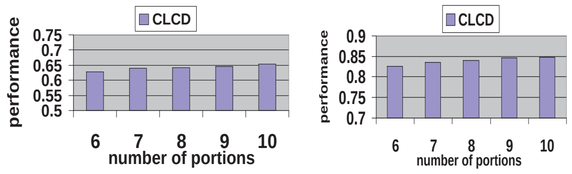
streams; consequently, the ensemble classifier becomes less accurate for predicting the coming data. However, CLCD decreases more slowly compared with soft-LELC, LELC and TSVM. This is because CLCD can identify random concept drift, so that CLCD offers a more accurate classifier.

In CLCD, we split each chunks into a number of portions, CLCD can accurately determine the concept drift if the concept drift occurs between the portions; if concept drift occurs within the portion of chunk, CLCD can just vaguely detect the concept drift and also perform better than the existing methods.

We set concept drift $r = 30\%$ and about 30% of the positive examples are labeled for the following experiments.

Figures 3 illustrates the performance comparison under different chunk sizes. We discover that the performance of these methods improves as the chunk size increases. In general, a large size chunk generally contains more data distribution information and offers a more accurate classifier. We further find that CLCD outperforms others at all times.

Figure 4 shows the performance comparison under different percentages of labeled examples with $p = 20\%$, $p = 30\%$ and $p = 40\%$ where $r = 30\%$ and chunk size equals to 400. We observe that for a specific p value, CLCD consistently outperforms others. When comparing four methods across different percentages of labeled examples, all four



(a) Newsgroup-20 (b) Web-KB

Figure 6: Sensitivity to different numbers of portions.

methods receive improvements as the label data increases. This indicates that providing a sufficient number of labeled samples will offer a more accurate classifier.

5.5 Efficiency Comparison Figure 5 shows the average running time of fifty chunks for each method with respect to different chunk size. It is found that CLCD takes the most time compared with LELC and soft-LELC, and it is consistent to the computational analysis in section section 4.3.3. LELC and soft-LELC take similar time, since their computational complexity is the same, as discussed in section 4.3.3. Further, TSVM takes more time than LELC, since TSVM constructs a transfer learning-based SVM for each chunk, while CLCD just constructs a transfer learning classifier for the chunks where concept drift occurs.

5.6 Sensitivity to Different Portion Number Figure 6 shows the sensitivity of the CLCD method to different numbers of portions from 6 to 10, in which chunk size is 400. Since we separate the chunks into smaller portions to determine concept drift, our *core vocabulary-based criteria* vaguely determine the change point. In general, if we split each chunk into more portions where each portion contains a number of samples, the predicted change point should be more accurate. From the two Figures, it is clear that the performance of CLCD is improved a little bit with the increase of portion size l and CLCD is only slightly sensitive to the portion number.

6 Conclusions

This paper has proposed an effective approach, named CLCD, to identify random concept drift in positive and unlabeled textual data stream environment. We put forward soft-LELC, on top of the LELC method, to extract representative examples and assign confidence scores for them; we then introduce a transfer learning-based SVM to build a predictive classifier by incorporating the representative examples and their confidence scores into learning. Extensive experiments have shown that CLCD can capture random concept drift and outperform state-of-the-art methods.

7 Acknowledgment

This work is supported in part by US NSF through grants IIS-0905215, CNS-1115234, IIS-0914934, DBI-0960443, and OISE-1129076, US Department of Army through grant W911NF-12-1-0066, Google Mobile 2014 Program, HUAWEI and KAU grants, Natural Science Foundation of China (61070033, 61203280, 61202270), Natural Science Foundation of Guangdong province (9251009001000005, S2011040004187, S2012040007078), Specialized Research Fund for the Doctoral Program of Higher Education (20124420120004), Australian Research Council Discovery Grant (DP1096218, DP130102691) and ARC Linkage Grant (LP100200774 and LP120100566).

References

[1] X. L. Li, P. S. Yu, B. Liu, and S. K. NG. Positive unlabeled learning for data stream classification. *SDM*, 2009.

[2] G. P. C. Fung, J. X. Yu, H. Lu, and P. S. Yu. Text classification without negative examples revisit. *TKDE*, 18:6–20, 2006.

[3] B. Liu, W. S. Lee, P. S. Yu, and X. Li. Partially supervised classification of text documents. *ICML*, 2002.

[4] H. Yu, J. Han, and K. C. C. Chang. Pebl: web page classification without negative examples. *TKDE*, 16(1):70–81, 2004.

[5] X. Li, B. Liu, and S. K. NG. Learning to classify documents with only a small positive training set. *ECML*, 2007.

[6] K. Zhou, G. Xue, and Q. Yang. Learning with positive and unlabeled exmples using topic-sensitive plsa. *TKDE*, 22(1):28–29, 2010.

[7] X. Li and B. Liu. Learning to classify texts using positive and unlabeled data. *IJCAI*, 2003.

[8] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu. Building text classifiers using positive and unlabeled examples. *ICDM*, 2003.

[9] Y. Xiao, B. Liu, J. Yin, L. Cao, C. Zhang, and Z. Hao. Similarity-based approach for positive and unlabeled learning. *IJCAI*, 2011.

[10] H. Wang, W Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. *KDD*, 2003.

[11] C. C. Aggarwal and P. S. Yu. Locust: An online analytical processing framework for high dimensional classification of data streams. *ICDE*, 2008.

[12] Y. Zhang, X. Li, and M. Orlowska. One-class classification of text streams with concept drift. *ICDM workshop*, 2008.

[13] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. *KDD*, 2001.

[14] P. Zhang, X. Zhu, and G. Li. Mining data streams with labeled and unlabeled training examples. *ICDM*, 2009.

[15] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.

[16] C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. *ACM SIGIR*, 1994.

[17] T. Evgeniou and M. Pontil. Regularized multi-task learning. *KDD*, 2004.

[18] V. Vapnik. Statistical learning theory. *Springer*, 1998.

[19] Gama J., Rocha R., and Medas P. Accurate decision trees for mining high-speed data streams. *KDD*, 2003.

[20] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.