Computing and Informatics, Vol. 41, 2022, 1260–1283, doi: 10.31577/cai_2022_5_1260

INCREMENTAL LEARNING METHOD FOR DATA WITH DELAYED LABELS

Haoran Gao, Zhijun Ding

The MOE Key Laboratory of Embedded System and Service Computation
Tongji University
Shanghai, 201804, China
e-mail: `ghrslyx@126.com, zhijun_ding@outlook.com`

Meiqin Pan*

School of Business and Management
Shanghai International Studies University
Shanghai, 200083, China
e-mail: `panmqin@sina.com`

Abstract. Most research on machine learning tasks relies on the availability of true labels immediately after making a prediction. However, in many cases, the ground truth labels become available with a non-negligible delay. In general, delayed labels create two problems. First, labelled data is insufficient because the label for each data chunk will be obtained multiple times. Second, there remains a problem of concept drift due to the long period of data. In this work, we propose a novel incremental ensemble learning when delayed labels occur. First, we build a sliding time window to preserve the historical data. Then we train an adaptive classifier by labelled data in the sliding time window. It is worth noting that we improve the TrAdaBoost to expand the data of the latest moment when building an adaptive classifier. It can correctly distinguish the wrong types of source domain sample classification. Finally, we integrate the various classifiers to make predictions. We apply our algorithms to synthetic and real credit scoring datasets. The experiment results indicate our algorithms have superiority in delayed labelling setting.

* Corresponding author

Keywords: Delayed labels, transfer learning, concept drift, incremental learning, credit scoring

Mathematics Subject Classification 2010: 68T10, 68U35

1 INTRODUCTION

With the rapid development of cloud computing, big data, and the Internet of Things, an increasing amount of data has been produced in the past few years in more and more industries. In the field of industry, data is often generated continuously with the development of business. Therefore, we often use machine learning to make analyses and predictions about the future using example data or experience. In many cases, a large portion of the research on machine learning tasks relies on the availability of true labels immediately. However, there are many real applications in which the labels are available with some delay. At time step t, the data denotes as $d_t = (x_t, y_t)$, where x_t is the feature vector, y_t is the label. Assuming that at time t only the feature vector x_t arrives, then the label y_t will arrive at time $t + \Delta t$. This phenomenon is called delayed labels [1, 2]. For example, in credit scoring [3], the objective is to predict whether a customer will default. But the true labels can be available within a few months or a few years after making a prediction. In fraud detection [4], the actual labels of the predicted examples are not available immediately. Therefore, delayed labels have emerged as one of the key concerns among researchers.

Assuming that there is a time delay varying from 0 to Δt for each sample. The model needs to predict the incoming data chunk of the next time. If we solve the problem of delayed labels with some of the existing algorithms, two problems will inevitably be encountered. First, each sample has a time delay varying from 0 to Δt at the same time. There remains a problem of concept drift due to the long period of data [5]. Second, the label for each data chunk will be obtained multiple times. Therefore, whether historical moment or the latest moment, labelled data is insufficient. The current study has tried to address these problems.

Kuncheva and Sanchez [6] explored whether the classifier should employ the unlabelled data to solve the problem of insufficient labelled data when the true labels come with a delay. This work chose to study IB2 and IB3 algorithms, which are the online nearest neighbour classifiers. Experiments show that IB2 benefits from unlabelled data, but IB3 does not. Therefore, this article did not make a conclusion about whether the classifier should employ the unlabelled data in delayed labelling setting. Krempl and Hofer [7] proposed drift models and discussed an exemplary adaptive learning algorithm. This work learned a mixture of labelled Gaussian as an adaptive learning strategy. But this approach required assumptions about the type of drift and formalized these assumptions. It is difficult to know about the type of drift in real situations. Dyer et al. [8] proposed the COMPacted Object Sample Extraction (COMPOSE) framework. This approach combined initial labels with new unlabelled data to train a semi-supervised classifier and maintained a compacted geometric center of each class distribution and adapt these centers by using new unlabelled data. Further, Frederickson and Polikar [9] proposed a variant extension of COMPOSE for handling scenarios with class-imbalance. But these articles proposed a framework in such an environment, where no labelled data are ever received after initialization. It does not match the most cases of delayed labels. Plasse and Adams [3] presented a classifier in delayed labelling setting to solve the long period of labelled data. This work introduced forgetting factors to reduce the weight of historical data to optimize streaming linear discriminant analysis (LDA). But this article did not explore whether concept drift occurs. If concept drift does not occur, the historical data can be valuable potentially. Therefore, reducing the weight of historical data blindly may harm the model performance. Grzenda et al. [2] proposed a novel evaluation method when delayed labels take place. This article refined model evaluation before the corresponding label arrived. The time of waiting for the labels is decomposed into subperiods. The evaluation of performance measures for every subperiod is made to analyze the changes in the performance measures between the initial predictions and when the labels are received. Further, Grzenda et al. [10] proposed the intermediate performance measures which extend the initial and the test-then-train performance. However, these works did not propose a novel classification when delayed labels take place. Pham et al. [11] examined the influence of delayed labels for Active Learning algorithms and proposed strategies to solve the problems. This work forgot outdated information and simulated the delayed labels to get more utility estimates. However, this article did not consider the influence of concept drift.

In conclusion, it is necessary to solve the two problems with the delayed labels. In this paper, we propose a novel incremental ensemble learning when delayed labels take place. When a new data chunk arrives, data of which true label is available, which means the labelled data, is used to train a classifier as a part of the ensemble model. In order to solve the problem of insufficient data, we need to integrate the historical labelled data with the latest labelled data when building a classifier. For this purpose, we build a sliding time window to preserve the historical data. Let T denote the length of the sliding time window. Data will be added to the sliding time window when their true labels become available. When the data chunk of the next time arrives, we need to train a classifier using labelled samples in the sliding time window for classification. On the other hand, delayed labels cause a problem of concept drift due to the long time intervals of data. Therefore, we need to use a drift detection algorithm to confirm if concept drift occurs in the sliding time window. If concept drift does not occur in the sliding time window, all data in the sliding time window will be directly used to build the classifier. Otherwise, all data in the sliding time window will be divided into two parts. One is the target domain where data keeps the latest distribution; the other is the source domain. Considering that the latest labelled data is insufficient and concept drift could only occur within some specific regions, we proposed an improved TrAdaBoost based on KNN to reuse the source domain data. Compared with the TrAdaBoost [12], our model can distinguish the error types of source data classification by judging whether the distribution of source domain changed. Some data in the source domain can be reused to expand the latest data to improve the performance of classification. In summary, the specific contributions of this work are as follows:

1. We propose a novel incremental ensemble learning when delayed labels take place. Our algorithms can minimize the impact of insufficient data and concept drift in delayed labelling setting by building a sliding time window, constructing adaptive classifiers and integrating the various classifiers.
2. We propose an improved TrAdaBoost based on KNN, which fully distinguishes the error types of source data classification. In this way, the problem of insufficient data at the latest moment is reduced in delayed labelling setting.
3. We apply our algorithms in credit scoring. The experiments indicate our algorithms have superiority.

2 PROPOSED METHOD

In this section, we first give the problem statement of delayed labels in Section 2.1. Then, in Section 2.2, the overall model framework is introduced. Finally, we will introduce building the sliding time window, constructing adaptive classifiers and integrating the classifiers in Sections 2.3, 2.4 and 2.5.

2.1 The Problem Statement of Delayed Labels

Firstly, we define the incremental data problem in delayed labels. The whole data comes into the system as a series of data chunks $D = \{D_1, D_2, \dots, D_t, \dots\}$, where D_t is the data chunk at time t. D_t can be expressed as $D_t = \{d_{(t,j)}\}_{j=1}^{N_t}$ with N_t samples. Hence, $d_{(t,j)} = (x_{(t,j)}, y_{(t,j)})$ is the j^{th} sample in D_t, where $x_{(t,j)} \in X$ is an input sample feature vector in the feature space X and $y_{(t,j)}$ is the label associated with $x_{(t,j)}$.

In order to reflect that different samples have different delayed times, we denote $\Delta\tilde{t}_{(t,j)}$ as a delayed period about the label for $d_{(t,j)}$. It indicates that the label for $d_{(t,j)}$ will arrive at time $t + \Delta\tilde{t}_{(t,j)}$. Let $[0, \Delta\tilde{t}_{max})$ be the value range of the delayed period, where 0 means that the label is available immediately after prediction and $\Delta\tilde{t}_{max} - 1$ is the maximum delayed time. The most commonly used symbols, notations and variables in the paper are summarized in Table 1.

2.2 Model Framework

This paper aims to solve the problem of concept drift in delayed label scenarios and the insufficient data caused by delayed labels. Therefore, we offer a novel incremental ensemble learning. We build a sliding time window to preserve the historical data

Symbol	Description
D_t	the data chunk at time t
N_t	the number of samples in D_t
$d_{(t,j)}$	the j^{th} sample in D_t
$x_{(t,j)}, y_{(t,j)}$	an input sample feature vector and the label for $d_{(t,j)}$
X	feature space
$\Delta\tilde{t}_{(t,j)}$	a delayed period about the label for $d_{(t,j)}$
$\Delta\tilde{t}_{max}$	the supremum of a delayed period about the label
L^t	the labelled data chunk which arrives at time t
$C^t_{(t-k)}$	the labelled data in data chunk D_{t-k} which arrives at time t
W^t	the sliding time window at time t
D^t_{t-k}	the labelled data in D_{t-k} before time $t-1$
T	the length of the sliding time window
μ_t	the average value of the sample features in D_t
σ_t	the standard deviation of the sample features in D_t
X^s	source domain
X^e	target domain
acc_{source}	the final average accuracy in the source domain samples
acc_{target}	the final average accuracy in the target domain samples
(x^s_b, y^s_b)	the b^{th} sample of X^s
(x^e_l, y^e_l)	the l^{th} sample of X^e
$knn^s(x^s_b)$	the number of sample labels among K nearest samples in source domain
$knn^e(x^s_b)$	the number of sample labels among K nearest samples in target domain
$\text{sim}(x^s_b)$	the sample distribution similarity of x^s_b
η	the threshold of sample distribution similarity
K	the number of nearest neighbour
w^f_b	the weight of source domain sample(x^s_b, y^s_b)
lr_1	the learning rate of weight updating formula when distribution is different
lr_2	the learning rate of weight updating formula when distribution is the same
$h_f(x^s_b)$	the classification result of x^s_b by the classifier built in the f^{th} iteration
ε_f	the error rate of the classifier built in the f^{th} iteration on X^e
$\bar{f}$	number of classifiers we selected
F	maximum iteration
$H^{(t)}$	the classifier set at time t
$WE^{(t)}$	the classifier weight set at time t
γ	the weight adjustment coefficient
H_q	the q^{th} classifier in classifier set
we^t_q	the weight of H_q at time t
θ^t_q	the evaluation performance of the classifier H_q on X^e
θ	the threshold of classifier weight

Table 1. The summary of used symbols, notations and variables in this paper

firstly. Then we create an adaptive classifier by the labelled data in the sliding time window. And we integrate the classifiers finally. The whole structure of the proposed model is shown in Figure 1. The overall model framework can be divided into three stages.

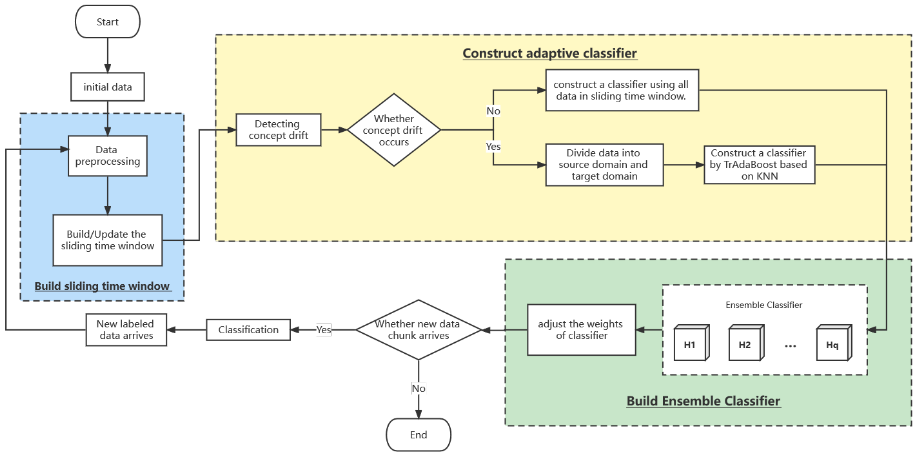
Start
initial data
Data preprocessing
Build/Update the sliding time window
Build sliding time window
Construct adaptive classifier
Detecting concept drift
Whether concept drift occurs
No
Yes
construct a classifier using all data in sliding time window.
Divide data into source domain and target domain
Construct a classifier by TrAdaBoost based on KNN
Ensemble Classifier
H1
H2
...
Hq
Build Ensemble Classifier
adjust the weights of classifier
Whether new data chunk arrives
Yes
No
Classification
New labeled data arrives
End

Figure 1. The overall model framework

- The first stage is to build a sliding time window to keep the historical data of the past T times. If new labelled data arrives, we will add it to the sliding time window.
- The second stage is to construct an adaptive classifier. First, we need to use a drift detection algorithm to confirm whether concept drift occurs between two adjacent data chunks in a sliding time window. If concept drift does not take place, all data in the sliding time window will be directly used to construct a classifier. Otherwise, we can build a classifier by transfer learning.
- The third stage is to build an ensemble classifier. First, we choose a diverse set of classifiers to integrate. Then, the ensemble classifier will predict the incoming data when we receive a new data chunk.

2.3 Building Sliding Time Window

In order to solve the problem of insufficient data in delayed label scenarios, we build a sliding time window to preserve historical data. When true labels of data arrive, they will be added to the sliding time window according to the arrival time of the feature. It can minimize the problem of insufficient data at each moment.

We denote $L^t = \{C^t_{t-\Delta t_{max}+1}, \dots, C^t_{t-2}, C^t_{t-1}, C^t_t\}$ as the labelled data chunks which arrive at time t, where $C^t_{(t-k)}$ is labelled data in data chunk D_{t-k} which arrives at time t. Considering that the incoming labelled data is insufficient at the latest time, we need to integrate the historical labelled samples and the latest time labelled samples. To this end, we build a sliding time window of length T to preserve the historical labelled data of the past T time.

We denote $W^{t-1} = \{D^{t-1}_{t-T}, \ldots, D^{t-1}_{t-2}, D^{t-1}_{t-1}\}$ as the sliding time window which constructs at time $t-1$, where D^{t-1}_{t-k} represents the labelled data in D_{t-k} before time $t-1$. When L^t arrives, we need to integrate it with W^{t-1}. Specifically, some new labelled data C^t_{t-k} arrives at time t. For each D^{t-1}_{t-k} in sliding time window, both D^{t-1}_{t-k} and C^t_{t-k} are subsets of D_{t-k}. The arrival time of the feature coincides between the two data chunks. Both D^{t-1}_{t-k} and C^t_{t-k} have the same concept. Therefore, we need to combine D^{t-1}_{t-k} with C^t_{t-k} to form D^t_{t-k}. D^t_{t-k} can be expressed as Equation (1).

$$D^t_{t-k} = D^{t-1}_{t-k} \cup C^t_{t-k}. \tag{1}$$

After integrating L^t with W^{t-1}, we need to delete D^{t-1}_{t-T} to ensure that the length of the sliding time window is T. Finally, the sliding time window W^t which constructed at time t is formed. The construction of our proposed sliding time window is shown in Figure 2.

In addition, considering that the order of magnitude of different dimensions in data is quite different, we need to standardize the data in the sliding time window according to the data chunk. The data standardization formula is as follows:

$$\overline{x_{(t,j)}} = \frac{x_{(t,j)} - \mu_t}{\sigma_t} \tag{2}$$

where $x_{(t,j)}$ is the j^{th} sample feature in D_t, μ_t is the average value of the sample features in D_t and σ_t is the standard deviation of the sample features in D_t.

2.4 Constructing Adaptive Classifier

At time t, the time window maintains data chunks from time $t-T+1$ to time t. We need to build a new classifier from the labelled data in the time window. Assuming that W^t is the sliding time window at time t, denoted as W^t. The time window at time t contains T data chunks with labelled samples $\{D^t_{(t-T+1)}, D^t_{(t-T+2)}, \ldots, D^t_t\}$. Concept drift might take place in these data chunks due to the long period of data. Therefore, we should use a drift detection algorithm to judge if concept drift occurs in the sliding time window. If concept drift does not take place, all data in the sliding time window will be directly used to construct a classifier. Otherwise, the labelled data which retains the latest concept in the time window W^t is denoted as X^e as the target domain and the remaining data is denoted as X^s as the source domain. Then we use the improved TrAdaBoost based on KNN to build a classifier. Therefore, constructing an adaptive classifier can be divided into two stages. The overall classifier construction is shown in Algorithm 1.

- The first stage is to use a drift detection algorithm to judge if concept drift occurs between two adjacent data chunks in a sliding time window.

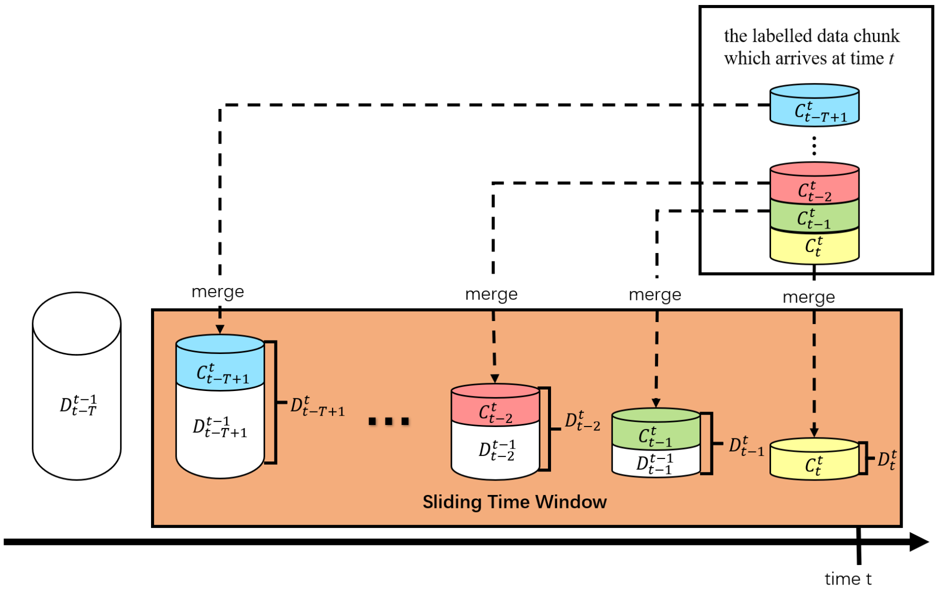

Figure 2. Building sliding time window

- The second stage is to build a new classifier. If concept drift occurs, we can build a classifier by transfer learning. Otherwise, all data in the sliding time window will be directly used to construct a classifier.

2.4.1 Detecting Concept Drift

We need to detect whether concept drift occurs between two adjacent data chunks in a sliding time window. Most concept drift detection methods first estimate the probability density functions of historical data and the latest data. Then, it evaluates their distance. The key is how to estimate the probability density functions. There have been related work using histograms density estimator to estimate the probability density functions. These algorithms adopt kdqtree [13], QuantTree [14] and EI-Kmeans [15] to build a histogram that partitions the feature space and evaluate the probability density function by estimating the number of samples in each feature partition. The robustness of the related algorithm has been verified. In this paper, we use EI-Kmeans for concept drift detection. EI-Kmeans is a novel space partitioning algorithm to address the problems caused by irregular partitions. Compared with Kmeans algorithm, EI-Kmeans begins by initializing the cluster centroids with a greedy equal intensity k-means initialization algorithm. It can search for the best centroids to reduce the randomness of algorithm. Second, in order to ensure that each cluster has enough samples for Pearson's chi-square test, EI-Kmeans

Algorithm 1 Building Adaptive Classifier

Input: W^t: sliding time window at time t; t: time

Output: $H_t(x)$: building model at time t; X^e: the labelled data which retains the latest concept

```
1: Initialize $X^e = []$, flag $= 0$
2: Set $X^e = D_t^t$
3: for $i \leftarrow 0$ to len$(W^t) - 2$ do
4:     Detect concept drift between $D_{(t-i-1)}^t$ and $D_{(t-i)}^t$
5:     if concept drift does not occur then
6:         $X^e = X^e \cup D_{(t-i-1)}^t$
7:     else
8:         $X^s = W^t - X^e$
9:         $H_t(x) =$ TrAdaBoost based on KNN$(X^s, X^e)$
10:        flag $= 1$
11:        break
12:    end if
13: end for
14: if flag $== 0$ then
15:     $H_t(x) =$ Create Classifier$(X^e)$
16: end if
17: return $H_t(x)$ and $X^e$.
```

uses an intensity-based amplify-shrink algorithm to unify the number of samples in the cluster. Compared with the kdqtree and QuantTree algorithms, EI-Kmeans builds a histogram by the clustering algorithm. It can reduce the randomness of partition results. The algorithm first adopts the EI-Kmeans clustering algorithm to partition the feature space. Then we apply Pearson's chi-square test to confirm if the number of data between the old feature partition and new feature partition has changed. If there is a significant change, concept drift has occurred. In our article, we separate the observations based on their labels and detect concept drift individually.

2.4.2 Building Classifier by Transfer Learning

When concept drift does not take place, all data in the sliding time window will be directly used to construct a classifier. When concept drift takes place, we divide the data in a sliding time window into two parts. Specifically, if concept drift occurs between $D_{(t-a)}^t$ and $D_{(t-a+1)}^t$ in W^t, we can divide W^t into two sets. We regard the labelled data which retains the latest concept in the time window as the target domain samples, denoted as $X^e = \{D_{(t-a+1)}^t \cup D_{(t-a+2)}^t \cup \cdots \cup D_t^t\}$. In this sample set, all the data chunks have the same distribution as D_t^t. It can be considered that these data chunks keep the latest distribution. Other data can be regarded as the source domain samples, denoted as $X^s = \{D_{(t-T+1)}^t \cup D_{(t-T+2)}^t \cup \cdots \cup D_{(t-a)}^t\}$.

In some cases, the target domain samples may be scarce. If the target domain samples are used to build a classifier, the classifier causes an overfitting result. Moreover, concept drift might only take place in specific areas, some source domain samples can still be reused. Therefore, we use transfer learning to reuse the source domain sample, which has the same distributions as the target domain.

Transfer AdaBoost, which is named TrAdaBoost [12], extends AdaBoost for transfer learning. For each weak learner, if source domain samples are wrongly classified, these samples will be dissimilar to the target domain sample. Therefore, we will decrease the weights of these source domain samples to weaken their impacts; if target domain samples are wrongly classified, we will gain the weight of these target domain samples to reduce the error of learner. However, there are two situations when source domain samples are wrongly classified. One is that the source domain samples have different distributions with target domain samples; the other is that the source domain samples have a similar distribution with target domain samples, but the classifier does not learn the feature of the sample. However, TradaBoost ignores the second case. Obviously, in the first situation, the source domain samples' weight should be reduced. But in the second situation, we need to increase its weight. Therefore, we proposed a TrAdaBoost based on KNN, which fully distinguishes the error types of source data classification and uses it in incremental models. Compared with traditional TrAdaBoost, our algorithm can more accurately determine the error types of the source domain sample. Compared with other classification algorithms, our algorithm can make use of the source domain sample and better extend the target domain sample by source domain sample.

We denote $X^s = \{(x_b^s, y_b^s)\}$ as source domain samples, where $b = 1, \ldots, m$. Target domain samples is denoted by $X^e = \{(x_l^e, y_l^e)\}$, where $l = 1, \ldots, n$. We need to build an ensemble classifier $H(x) : x \to y$ to optimize its performance.

First of all, we need to initialize the weights of the source domain samples and the target domain samples. TrAdaBoost allows users to define the initial weights of the samples. Some papers [16] believe that the source domain samples and the target domain samples need to set the same initial weights. We consider that when the distribution of source domain samples and target domain samples is quite different and the number of source domain samples is greater than the number of target domain samples, the classifier may wrongly bias towards source domain samples at first. In this case, we want to set lower initial weights for source domain samples. Therefore, we first divide the source domain sample set into z sub-sample sets on average. Each time we take $z - 1$ sub-sample sets separately to train the classifier and the remaining one sub-sample set to evaluate the accuracy. After z repetitions, the final average accuracy in the source domain samples will be denoted as acc_{source}. Similarly, the classifier is used to evaluate the samples in the target domain. The final average accuracy in the target domain samples will be denoted as acc_{target}. If there is a large gap between acc_{source} and acc_{target}, we should set a lower initial weight to the source domain samples. Therefore, the initial weight vector is denoted by $w^1 = (w_1^1, \dots, w_{n+m}^1)$, and w_i^1 can be acquired by Equation (3).

$$w_i^1 = \begin{cases} \frac{\log(acc_{target}/(1-acc_{target}))}{\log(acc_{source}/(1-acc_{source}))}, & i = 1, \dots, m, \\ 1, & i = m+1, \dots, m+n. \end{cases} \tag{3}$$

Since original TrAdaBoost cannot determine the cause of the wrong classification of the source domain samples, we need to determine whether the source domain sample's distribution is different from the samples in target domain before updating the weights of samples. For a sample in the source domain, if its nearest neighbor's label in source domain is quite different from its nearest neighbor's label in target domain, it is believed that the distribution of the sample has been changed. Obviously, we can use KNN model. The main idea of KNN is that given a sample, we can use certain neighbor measure to calculate the neighbor degrees to find the K nearest neighbor. Given a sample in the source domain, if the majority labels of its K closest samples change between the source domain and target domain, the distribution of the samples could be considered to change. Specifically, we first build two KNN models on the source and target domains. Given the source domain sample (x_b^s, y_b^s), we calculate the number of sample label y_b^s among the K nearest samples in the source domain, denoted as $knn^s(x_b^s)$. We also calculate the number of sample label y_b^s among the K nearest samples in the target domain, denoted as $knn^e(x_b^s)$. Finally, the sample distribution similarity of source domain can be denoted by Equation (4).

$$\text{sim}(x_b^s) = \frac{knn^e(x_b^s)}{knn^s(x_b^s)}. \tag{4}$$

Obviously, if $\text{sim}(x_b^s) < \eta$, it will be considered that the distribution of samples has changed greatly between the source domain and target domain. During updating the weights of samples, it is necessary to reduce the weight of the sample when source domain samples are wrongly classified. If $\text{sim}(x_b^s) \geq \eta$, it means that the source domain samples and the target domain samples have the same distribution. We need to increase the weight of the sample in this case.

After computing the similarity of source domain samples, we should improve the mechanism of updating the weights of samples. In the $(f+1)^{\text{th}}$ iteration, the weight of the source domain sample can be expressed by Equation (5).

$$w_b^{f+1} = \begin{cases} w_b^f \times lr_1 \times \beta^{\left|h_f(x_b^s) - y_b^s\right|}, & \text{sim}(x_b^s) < \eta, \\ w_b^f \times lr_2 \times \beta^{-\left|h_f(x_b^s) - y_b^s\right|}, & \text{sim}(x_b^s) \geq \eta, \end{cases} \tag{5}$$

where $\beta = \frac{\varepsilon_f}{1-\varepsilon_f}$, lr_1 represents the learning rate of weight updating formula when the distribution is different, lr_2 represents the learning rate of weight updating formula when the distribution is the same, w_b^f is the weight of source domain sample (x_b^s, y_b^s) in f^{th} iteration, $h_f(x_b^s)$ is the classification result of x_b^s by the classifier constructed in the f^{th} iteration. The error rate of the classifier constructed in the f^{th} iteration on X^e can be expressed by Equation (6).

$$\varepsilon_f = \sum_{l=1}^{n} \frac{w_l^f \left| h_f(x_l^e) - y_l^e \right|}{\sum_{l=1}^{n} w_l^f} \tag{6}$$

where w_l^f is the weight of the target domain sample (x_l^e, y_l^e) in the f^{th} iteration.

In the $(f+1)^{\text{th}}$ iteration, the weight of the target domain sample (x_l^e, y_l^e) can be expressed by Equation (7).

$$w_l^{f+1} = w_l^f \times lr_2 \times \beta^{-\left| h_f(x_l^e) - y_l^e \right|}. \tag{7}$$

To sum up, when the source domain sample is misclassified, we should judge whether its distribution is different. The sample distribution similarity of source domain is calculated by Equation (4). If the similarity of a certain sample is less than the threshold η, it could be considered that the distribution of sample in the source domain is different from the distribution of sample in the target domain. We reduce the weight of this sample by Equation (5). Similarly, if they are similar, we will increase the weight of sample by Equation (5). When the samples in target domain are wrongly classified, we will increase the weight of sample by Equation (7).

Finally, we need to integrate the weak classifiers. We assume that a total of F weak classifiers are generated. The original paper of TrAdaBoost requires the last $\frac{F}{2}$ weak classifiers for integration. However, in our test, it does not always achieve the best results. The reason is probably that the later weak classifiers are given too much weight to the misclassified samples in the target domain. It causes that the classifier has a harmful effect by noise or outliers. Therefore, we set a hyperparameter $\bar{f}$, and it means that we select the last $\bar{f}$ classifiers for ensemble. The hyperparameter $\bar{f}$ can be adjusted according to the performance of the model. We use the weighted voting method for ensemble. The number of votes of each weak classifier is multiplied by the weight, and the weighted number of votes of each category is summed up. The category corresponding to the maximum value is the final category. The final robust classifier is as follows:

$$H(x) = \operatorname{sign}\left(\sum_{f=F-\bar{f}}^{F} \alpha_f h_f(x) \right) \tag{8}$$

where sign() is sign function, if $z > 0$, $\operatorname{sign}(z) = 1$; otherwise, $\operatorname{sign}(z) = 0$; and α_f represents the weight of the weak classifier generated by the f^{th} iteration, α_f can be expressed as:

$$\alpha_f = \frac{1}{2} \log \frac{1 - \varepsilon_f}{\varepsilon_f}. \tag{9}$$

Adaptive classifier construction is shown in Algorithm 2.

Algorithm 2 TrAdaBoost based on KNN

```
Input: $X^s$: source domain samples; $X^e$: target domain samples; F: maximum iteration; $\bar{f}$: number of classifiers we selected; $K$: the number of nearest neighbors
Output: $H\left(x\right)$: the final classifier
 1: The initial weight vector $w^1 = (w_1^1, \ldots, w_{n+m}^1)$ and $w_i^1$ can be acquired by Equation (3)
 2: Construct two KNN model on source domain and target domain, calculate the sample distribution similarity of source domain by Equation (4)
 3: while $f \leq F$ do
 4:     Set $p^f = w^f / \sum_{i=1}^{n+m} w_i^f$
 5:     Train a classifier by $X^s$ and $X^e$: $h_f(x) : X \to Y$
 6:     Calculate the error rate of the classifier $h_f(x)$ on $X^e$ by Equation (6)
 7:     Calculate the weight of the weak classifier $h_f(x)$ by Equation (9)
 8:     Set $\beta = \varepsilon_f / (1 - \varepsilon_f)$
 9:     if $x \in X^s$ then
10:         Update the weight by Equation (5)
11:     else
12:         Update the weight by Equation (7)
13:     end if
14: end while
15: return classifier $H(x)$ by Equation (8)
```

2.5 Building Ensemble Classifier

At time $t-1$, we maintain p classifiers trained on the data chunks from 1 to $t-1$ to form a classifier set, denoted as $H^{(t-1)} = \{H_1, H_2, \ldots, H_p\}$. The classifier weight set is denoted as $WE^{(t-1)} = \{we_1^{(t-1)}, we_2^{(t-1)}, \ldots, we_p^{(t-1)}\}$, where $we_q^{(t-1)}$ is the weight of H_q at time $t-1$. After learning a new classifier H on W^t, we need to adjust the weights of the p classifiers in $H^{(t-1)}$. The adjustment strategy is based on the evaluation performance of each base classifier on the target domain samples X^e. X^e is the labelled data that retains the latest concept in the time window. Since concept drift does not occur in X^e, it can be considered that it represents the latest concept. We denote θ_q^t as the evaluation performance of the classifier H_q on X^e at time t. Evaluation performance can be calculated by any error function according to the dataset, such as error rate. The weight of H_q can be expressed by Equation (10) at time t.

$$we_q^t = (1 - \theta_q^t) we_q^{(t-1)\gamma} \tag{10}$$

where γ is the weight adjustment coefficient, $\gamma \in [0, 1]$. When γ is 0, the weight of the classifier is only related to the evaluation performance. When γ is 1, the weight of the classifier at time t is related to the evaluation and the weight at time $t-1$. In addition, in order to avoid the unlimited addition of base classifiers, the classifiers with weight less than threshold θ are removed. After adjusting the weight, we should merge the newly trained classifier H with $H^{(t-1)}$ to form H^t and set the weight of the classifier H to 1. Finally, the new ensemble model H^t predicts the incoming data chunk D_{t+1}.

Algorithm 3 Incremental Ensemble Learning

Input: $D = \{D_1, D_2, \dots, D_t, \dots\}$: a series of data chunks; H^t: a classifier set at time t; θ: threshold; WE^t: the classifier weight set at time t; γ: the weight adjustment coefficient

Output: prediction results

```
1: Initialize $t = 1$; $H^0 = []$; $WE^0 = []$; $W^0 = []$;
2: for $t = 1, 2, \dots$ do
3:     Normalize $D^t$ by Equation (11)
4:     Predict $D^t$ by $H^{(t-1)}$
5:     Normalize $D^t$ by Equation (2)
6:     for $k \leftarrow 0$ to $len(W^{t-1}) - 1$ do
7:         Update sliding time window by Equation (1)
8:     end for
9:     if $len(W^t) > T$ then
10:        delete($W^t$,$D^{(t-1)}_{(t-T)}$)
11:    end if
12:    if $len(W^t) > 0$ then
13:        $H_t, X^e$ = building adaptive classifier($W^t, t$)
14:        if $len(H^{(t-1)}) > 0$ then
15:            for $q \leftarrow 1$ to $len(H^{(t-1)})$ do
16:                Update the weight of base classifier by Equation (10)
17:                if $we^t_q < \theta$ then
18:                    Remove classifiers with weights less than $\theta$
19:                end if
20:            end for
21:        end if
22:        $H^t = H^{(t-1)} \cup H_t$; $we^t_t = 1$; $WE^t = WE^{(t-1)} \cup we^t_t$
23:    end if
24: end for
25: return prediction results
```

When we make a prediction to the data chunk D_{t+1} at time $t + 1$, we need to standardize samples in D_{t+1} to eliminate the influence of different orders of magnitude of different dimensions in data. Considering that we cannot get the whole data chunk D_{t+1} immediately at time $t + 1$, we need to retain the mean and standard deviation in the data chunk D_t to standardize. The standardization formula is as follows:

$$\overline{x_{(t+1,j)}} = \frac{x_{(t+1,j)} - \mu_t}{\sigma_t} \tag{11}$$

where $x_{(t+1,j)}$ is the j^{th} sample feature in D_{t+1}, μ_t is the average value of the sample features in D_t and σ_t is the standard deviation of the sample features in D_t.

After making a prediction on D_{t+1}, we standardize D_{t+1} with the mean and standard deviation in D_{t+1} and update the time window through the sliding time window updating mechanism proposed in Section 2.3. The framework of the entire model is shown in Algorithm 3.

3 EXPERIMENT

In this section, we first introduce experiments on synthetic datasets in Section 3.1. In Section 3.2, experiments on real credit scoring datasets are introduced.

3.1 Experiments on Synthetic Datasets

3.1.1 Synthetic Datasets

In order to evaluate the effectiveness of the algorithm in this paper, six synthetic datasets are used to evaluate the performance at first. The samples of these six datasets are marked with the generation time of the sample features, but the delay time of the label is not given. So we manually construct the delay time of label on these six datasets. Specifically, for each synthetic dataset, we assume that the label delay time of each sample in $[0, \Delta\tilde{t}_{max})$ is randomly generated. The six datasets are described as follows:

SEA [17]: This dataset contains 50 000 samples. It is generated using three attributes, which only the two first attributes are relevant. The classification is done using $\alpha_1 + \alpha_2 = \beta$, where β is a threshold value. In this paper, each data chunk is set to contain 1 000 samples. We set $\Delta\tilde{t}_{max} = 10$ and $\Delta\tilde{t}_{max} = 5$.

Hyperplane [18]: This dataset contains 50 000 samples. It is a two-dimensional dataset which simulates concept drift by changing the rotation angle of the decision hyperplane. In this paper, each data chunk is set to contain 1 000 samples. We set $\Delta\tilde{t}_{max} = 10$ and $\Delta\tilde{t}_{max} = 5$.

Agrawal [19]: This dataset contains 50 000 samples. It is generated using nine attributes, six numeric and three categorical. These attributes describe hypothetical loan applications to simulate whether a loan should be approved. In this paper, each data chunk is set to contain 1 000 samples. We set $\Delta\tilde{t}_{max} = 10$ and $\Delta\tilde{t}_{max} = 5$.

RBF: This dataset contains 50 000 samples. It is generated using ten numerical attributes. It contains two classes. Each class has a different class center. Every sample has a certain offset from the category center, which gradually shifts over time. In this paper, each data chunk is set to contain 1 000 samples. We set $\Delta\tilde{t}_{max} = 10$ and $\Delta\tilde{t}_{max} = 5$.

CSDS3 [20]: This dataset is a credit scoring dataset provided by anonymous companies. It contains information about 97 thousand loans that were analyzed, approved and conceded by a large bank in Brazil. It includes loan requests from October 2013 to January 2015. In this dataset, we divide it into 16 data chunks at one-month intervals. Since the credit scoring dataset requires determining whether the user defaults within three months, we set $\Delta\tilde{t}_{max} = 4$.

Fraud: This dataset which is provided by the Kaggle platform requires determining whether fraudulent transactions have occurred in transactions. It includes transaction information from January 2019 to December 2020. In this dataset, we divide it into 24 data chunks at one-month intervals. We set $\Delta\tilde{t}_{max} = 5$.

Dataset	Samples	Features	Chunks
SEA	50 000	3	50
Hyperplane	50 000	2	50
Agrawal	50 000	9	50
RBF	50 000	10	50
CSDS3	97 226	152	16
Fraud	1 852 394	22	24

Table 2. Information of six datasets

3.1.2 Experiment Settings

First of all, this paper selects three classical chunk-based methods for solving the concept drift problem to compare with our model. The compared methods are as follows.

1. Learn++ [21]: This method first divides the data into multiple data chunks $\{D_1, D_2, \ldots, D_t, \ldots\}$. Each chunk is built with a classifier. When the latest data chunk D_t arrives, it trains a base classifier by D_{t-1} and adjusts the weights of each previous base classifier according to their performance and a time-decay function. Then it forms an ensemble classifier.
2. REA [22]: It is an earlier method to solve the concept drift of unbalanced data. When the latest chunk D_t arrives, this method adaptively pushes into D_{t-1} part of minority class examples received within $[1, t-2]$ to balance its skewed class distribution. These samples are merged with D_{t-1} for training.
3. DWMIL [23]: This method adopts incremental learning to solve the problem of concept drift. Its basic idea is similar to Learn++. However, the formula for adjusting the weight of the base classifier in DWMIL is different from Learn++. In addition, DWMIL deletes premature classifiers to avoid excessive model complexity.

The above three classical methods for solving the concept drift problem are applied to the situation without delayed labels. When D_{t-1} is predicted, the true label of D_{t-1} can be obtained immediately. Therefore, these methods use D_{t-1} to build classifiers to predict D_t. However, in the case of delayed labels, the true label of the data chunk cannot be obtained after making predictions. Therefore, it is necessary to improve the above three comparative methods. We build a classifier using L^{t-1} which is the labelled data chunk arriving at time $t-1$ to predict D_t.

Secondly, we use Accuracy and Area Under Curve (AUC) as the evaluation metrics. Specifically, we calculate the average Accuracy of each data chunk on the SEA, Hyperplane, RBF and Agrawal datasets. Accuracy is expressed as the ratio of the number of samples correctly classified to the total number of samples for a given dataset, and the formula is as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{12}$$

where TP is true positives, FP is false positives, FN is false negatives, and TN is true negatives.

For CSDS3 and Fraud datasets, we use AUC as the evaluation metric. Because these datasets are unbalanced, if we use Accuracy as the evaluation metric on these datasets, the test results might be inaccurate. AUC represents the area under the ROC curve. AUC has been widely used for measuring classification performance with imbalanced data distributions. Therefore, we calculate the average AUC of each data chunk. The formula of AUC is as follows:

$$\text{AUC} = \frac{\sum_{positives} r - \frac{n_{pos}(n_{pos}+1)}{2}}{n_{pos}n_{neg}} \tag{13}$$

where r is the probability, which the data is predicted to be a positive sample, n_{pos} is the number of positive samples in the dataset, and n_{neg} is the number of negative samples in the dataset.

Finally, for the sliding time window module, we set the length of the sliding time window $T = \Delta\tilde{t}_{max}$. In the adaptive classifier module, for the TrAdaBoost based on KNN, we set decision tree as the base classifier, the threshold $\eta = 0.7$, the maximum number of iterations $F = 50$, learning rate for weight update when the distribution is different $lr_1 = 0.5$, learning rate for weight update when the distribution is the same $lr_2 = 0.3$, the number of nearest neighbors $K = 20$ in KNN. For different datasets, due to different degrees of concept drift, there will be some differences in performance when the number of retained base classifiers is different. We set the number of base classifiers $\bar{f}$ from 26 to 50 with a steps size of 5 and select the optimal parameters. If concept drift does not occur, we use AdaBoost as a classifier and use decision tree as a base classifier. For the ensemble classifier module, we set the weight adjustment factor $\gamma = 0.5$, the detection threshold $\theta = 0.6$. We use AdaBoost as a classifier for comparative experiments.

3.1.3 Experiment Results on Synthetic Datasets

We first conduct comparative experiments on the SEA, Hyperplane, Agrawal, and RBF datasets. Table 3 shows the comparison between our method and the other three methods on these datasets. It can be seen from the experiment results that our method can more effectively solve the problems of insufficient samples and concept drift caused by delayed labels. In addition, we also conduct ablation experiments

Dataset	Delayed Time	Our Method	DWMIL	Learn++	REA
SEA	$\Delta\tilde{t}_{max} = 10$	**86.41 %**	84.25 %	83.55 %	83.33 %
	$\Delta\tilde{t}_{max} = 5$	**86.98 %**	85.94 %	84.54 %	84.03 %
Hyperplane	$\Delta\tilde{t}_{max} = 10$	**96.77 %**	96.18 %	94.48 %	91.49 %
	$\Delta\tilde{t}_{max} = 5$	**97.16 %**	97.03 %	95.58 %	93.94 %
Agrawal	$\Delta\tilde{t}_{max} = 10$	**89.47 %**	86.22 %	83.24 %	73.27 %
	$\Delta\tilde{t}_{max} = 5$	**91.07 %**	89.35 %	87.93 %	78.30 %
RBF	$\Delta\tilde{t}_{max} = 10$	**91.58 %**	89.31 %	85.29 %	79.70 %
	$\Delta\tilde{t}_{max} = 5$	**92.53 %**	90.94 %	88.06 %	81.70 %

Table 3. Accuracy performance on the SEA, Hyperplane, Agrawal and RBF datasets in the comparison experiment

Dataset	Delayed Time	Our Method	Without Transfer Learning	TrAdaBoost
SEA	$\Delta\tilde{t}_{max} = 10$	**86.41 %**	85.79 %	85.88 %
	$\Delta\tilde{t}_{max} = 5$	**86.98 %**	86.33 %	86.48 %
Hyperplane	$\Delta\tilde{t}_{max} = 10$	**96.77 %**	96.07 %	96.07 %
	$\Delta\tilde{t}_{max} = 5$	**97.16 %**	97.08 %	**97.16 %**
Agrawal	$\Delta\tilde{t}_{max} = 10$	**89.47 %**	89.20 %	88.36 %
	$\Delta\tilde{t}_{max} = 5$	**91.07 %**	89.81 %	90.91 %
RBF	$\Delta\tilde{t}_{max} = 10$	**91.58 %**	90.92 %	91.20 %
	$\Delta\tilde{t}_{max} = 5$	**92.53 %**	92.02 %	92.15 %

Table 4. Accuracy performance on the SEA, Hyperplane, Agrawal and RBF datasets in the ablation experiment

to verify the effectiveness of TrAdaBoost based on KNN. The ablation experiment includes two methods to compare with our model. Only target domain data was used to construct classifiers without transfer learning, and the TrAdaBoost method was used to construct classifiers. The experiment results on the SEA, Hyperplane, Agrawal and RBF datasets are shown in Table 4. From Table 4, we can see that transfer learning can achieve better results compared with the method which constructs classifiers without transfer learning due to insufficient sample. Compared with TrAdaBoost, the transfer learning proposed in this paper can correctly distinguish the types of classification errors and achieve better results. It shows that our method can better handle the problem of insufficient samples and concept drift caused by delayed labels.

Then we conduct comparative experiments on CSDS3 and Fraud datasets. The comparison results on AUC are shown in Table 5. Meanwhile, ablation experiments on CSDS3 and Fraud datasets are also conducted. The comparison results on AUC are shown in Table 6. It shows that our method is better than other methods on AUC in CSDS3 dataset, but DWMIL is better than other methods in Fraud. In the ablation experiment, the method which constructs classifiers without transfer learning is better than other methods. The possible reason is that the concept drift in Fraud dataset is weak or does not occur, so the method we proposed in this paper is not significantly different from other methods. Our method is generally superior to other comparative experimental methods on six synthetic datasets.

Dataset	Delayed Time	Our Method	DWMIL	Learn++	REA
CSDS3	$\Delta\tilde{t}_{max} = 4$	**82.35 %**	81.79 %	81.86 %	81.89 %
Fraud	$\Delta\tilde{t}_{max} = 5$	98.11 %	**98.29 %**	98.12 %	97.83 %

Table 5. AUC performance on the CSDS3 and Fraud datasets in the comparison experiment

Dataset	Delayed Time	Our Method	Without Transfer Learning	TrAdaBoost
CSDS3	$\Delta\tilde{t}_{max} = 4$	**82.35 %**	80.29 %	81.69 %
Fraud	$\Delta\tilde{t}_{max} = 5$	98.11 %	**98.37 %**	97.48 %

Table 6. AUC performance on the CSDS3 and Fraud datasets in the ablation experiment

3.2 Experiments on Real Credit Scoring Datasets

3.2.1 Real Datasets

In this section, we choose two real credit scoring datasets for experiments. These datasets give the generation time of sample features and give the real delay time of each sample label. Specifically, the delay time of each sample label can be obtained by subtracting the time of the loan application from the time of the last loan submission. The two datasets are described as follows.

PPDAI: PPDAI is a very representative enterprise among Chinese Internet finance companies. This dataset, which includes 10 % sampling data of all credit targets from January 2015 to January 2017, is the real business data provided by PPDAI online lending platform. This dataset needs to determine whether the user is default. We divide it into a total of 24 data chunks at one-month intervals. We select the data in this dataset with a loan period of less than or equal to 18 months for testing. We set $\Delta\tilde{t}_{max} = 19$ in PPDAI dataset.

LendingClub: LendingClub is the world's largest P2P Internet lending platform. This platform updates the lending data in real time. Therefore, in order to ensure that the data which is being repaid is not included in the dataset, we use the data in LendingClub from June 2007 to December 2016 and select the data with the loan period of 3 years. The total number of data in this dataset is 944 664. It is generated by 149 dimensions. Each sample represents the information of a customer.

We mark users who fully repay their loans as customers with good credit and those who default as customers with bad credit. In this dataset, we divide it into 115 data chunks at one-month intervals. We set $\Delta\tilde{t}_{max} = 37$.

Dataset	Samples	Features	Chunks
PPDAI	113 594	20	24
LendingClub	944 664	149	115

Table 7. Information of credit scoring datasets

3.2.2 Experiment Settings

The compared method we selected in this section is the same as the compared method in Section 3.1. For LendingClub and PPDAI real dataset, we select AUC as the evaluation metric since these datasets are imbalanced. We evaluate the dataset by calculating the average AUC on each data chunk. In addition, for PPDAI, the last six data chunks contain too much unlabelled data which are being repaid and do not know the true label. The evaluation results will be biased on the last six data chunks. So we adopt PPDAI data from the first 18 months to evaluate. For LendingClub, a qualified classifier cannot be built for evaluation due to the insufficient amount of data at the beginning, and we evaluate the LendingClub data of the last 36 months.

Finally, for the sliding time window module, we set the length of the sliding time window $T = \Delta\tilde{t}_{max}$. In the adaptive classifier module, for the TrAdaBoost based on KNN, we set decision tree as the base classifier, the threshold $\eta = 0.6$, the maximum number of iterations $F = 50$, learning rate for weight update when the distribution is different $lr_1 = 0.5$, learning rate for weight update when the distribution is the same $lr_2 = 0.3$, the number of nearest neighbors $K = 20$ in KNN. For different datasets, due to different degrees of concept drift, there will be some differences in performance when the number of retained base classifiers is different. We set the number of base classifiers $\bar{f}$ from 26 to 50 with a steps size of 5 and select the optimal parameters. If concept drift does not occur, we use AdaBoost as a classifier and use decision tree as a base classifier. For the ensemble classifier module, we set the weight adjustment factor $\gamma = 0$, the detection threshold $\theta = 0.5$. We use AdaBoost as a classifier for comparative experiments.

3.2.3 Experiment Results on Real Datasets

We conduct comparative experiments on LendingClub and PPDAI datasets. Table 8 shows the comparison between our method and the other three methods on these datasets. Meanwhile, we conduct ablation experiments on LendingClub and PPDAI datasets. The results are shown in Table 9. It can be seen from the experimental results that our method is superior to the other three methods. It shows that there are two advantages to our method.

1. The method in this paper solves the problem of insufficient samples in delayed labelling setting by expanding the number of samples. The experimental results show that it can achieve better results.
2. Our method can better solve the problem of concept drift in delayed labelling setting. When concept drift occurs, our method is adopted to reuse the data in the source domain.

Similarly, the ablation results demonstrate the effectiveness of the TrAdaBoost based on KNN.

Dataset	Delayed Time	Our Method	DWMIL	Learn++	REA
LendingClub	$\Delta\tilde{t}_{max} = 37$	**67.70 %**	65.76 %	66.97 %	67.10 %
PPDAI	$\Delta\tilde{t}_{max} = 19$	**64.30 %**	62.47 %	63.45 %	61.59 %

Table 8. AUC performance on the LendingClub and PPDAI datasets in the comparison experiment

Dataset	Delayed Time	Our Method	Without Transfer Learning	TrAdaBoost
LendingClub	$\Delta\tilde{t}_{max} = 37$	**67.70 %**	66.60 %	66.04 %
PPDAI	$\Delta\tilde{t}_{max} = 19$	**64.30 %**	62.02 %	63.55 %

Table 9. AUC performance on the LendingClub and PPDAI datasets in the ablation experiment

4 CONCLUSION

In this paper, we propose an incremental ensemble learning model when delayed labels take place. Our algorithms can minimize the impact of insufficient data and concept drift in delayed labels by building a sliding time window, constructing adaptive classifiers and integrating the various classifiers. In addition, we propose an improved TrAdaBoost based on KNN, which can more effectively determine the error type in the source domain. Our results show that our method can significantly improve the performance of the classifier. In summary, our method can better adapt to the scenario of delayed label.

There are also some shortcomings in our paper. First of all, some historical samples are stored when the classifier is built in this paper. It will lead to an increase the time complexity and space complexity. Therefore, ensuring performance while reducing time and space complexity is the key to our next research. Secondly, we do not take advantage of the unlabelled data generated at every moment. Applying the unlabelled data in our model is the key to our next research. We believe that such work can solve the delayed labelling problem more effectively.

Acknowledgement

This work is partially supported by the Shanghai "Science and Technology Innovation Action Development" High-tech Field Program under Grant 19511101300, and by the Shanghai International Studies University Development Program under Grant 2019114009.

REFERENCES

[1] Gomes, H. M.—Bifet, A.—Read, J.—Barddal, J. P.—Enembreck, F.—Pfharinger, B.—Holmes, G.—Abdessalem, T.: Adaptive Random Forests for Evolving Data Stream Classification. Machine Learning, Vol. 106, 2017, No. 9, pp. 1469–1495, doi: 10.1007/s10994-017-5642-8.

[2] Grzenda, M.—Gomes, H. M.—Bifet, A.: Delayed Labelling Evaluation for Data Streams. Data Mining and Knowledge Discovery, Vol. 34, 2020, No. 5, pp. 1237–1266, doi: 10.1007/s10618-019-00654-y.

[3] Plasse, J.—Adams, N.: Handling Delayed Labels in Temporally Evolving Data Streams. 2016 IEEE International Conference on Big Data (Big Data), 2016, pp. 2416–2424, doi: 10.1109/BigData.2016.7840877.

[4] Carcillo, F.—Dal Pozzolo, A.—Le Borgne, Y. A.—Caelen, O.—Mazzer, Y.—Bontempi, G.: SCARFF: A Scalable Framework for Streaming Credit Card Fraud Detection with Spark. Information Fusion, Vol. 41, 2018, pp. 182–194, doi: 10.1016/j.inffus.2017.09.005.

[5] Lu, J.—Liu, A.—Dong, F.—Gu, F.—Gama, J.—Zhang, G.: Learning Under Concept Drift: A Review. IEEE Transactions on Knowledge and Data Engineering, Vol. 31, 2019, No. 12, pp. 2346–2363, doi: 10.1109/TKDE.2018.2876857.

[6] Kuncheva, L. I.—Sánchez, J. S.: Nearest Neighbour Classifiers for Streaming Data with Delayed Labelling. 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 869–874, doi: 10.1109/ICDM.2008.33.

[7] Krempl, G.—Hofer, V.: Classification in Presence of Drift and Latency. 2011 IEEE 11th International Conference on Data Mining Workshops, 2011, pp. 596–603, doi: 10.1109/ICDMW.2011.47.

[8] Dyer, K. B.—Capo, R.—Polikar, R.: COMPOSE: A Semisupervised Learning Framework for Initially Labeled Nonstationary Streaming Data. IEEE Transactions on Neural Networks and Learning Systems, Vol. 25, 2014, No. 1, pp. 12–26, doi: 10.1109/TNNLS.2013.2277712.

[9] Frederickson, C.—Polikar, R.: Resampling Techniques for Learning Under Extreme Verification Latency with Class Imbalance. 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1–8, doi: 10.1109/IJCNN.2018.8489622.

[10] Grzenda, M.—Gomes, H. M.—Bifet, A.: Performance Measures for Evolving Predictions Under Delayed Labelling Classification. 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–8, doi: 10.1109/IJCNN48605.2020.9207256.

[11] Pham, T.—Kottke, D.—Krempl, G.—Sick, B.: Stream-Based Active Learning for Sliding Windows Under the Influence of Verification Latency. Machine Learning, Vol. 111, 2022, No. 6, pp. 2011–2036, doi: 10.1007/s10994-021-06099-z.

[12] Dai, W.—Yang, Q.—Xue, G. R.—Yu, Y.: Boosting for Transfer Learning. Proceedings of the 24th International Conference on Machine Learning (ICML 2007), 2007, pp. 193–200, doi: 10.1145/1273496.1273521.

[13] Dasu, T.—Krishnan, S.—Venkatasubramanian, S.—Yi, K.: An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams. Proceedings of the 38th Symposium on the Interface of Statistics, Computing Science, and Applications 2006 (Interface 2006), 2006.

[14] Boracchi, G.—Carrera, D.—Cervellera, C.—Maccio, D.: QuantTree: Histograms for Change Detection in Multivariate Data Streams. In: Dy, J., Krause, A. (Eds.): Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 80, 2018, pp. 639–648.

[15] Liu, A.—Lu, J.—Zhang, G.: Concept Drift Detection via Equal Intensity K-Means Space Partitioning. IEEE Transactions on Cybernetics, Vol. 51, 2021, No. 6, pp. 3198–3211, doi: 10.1109/TCYB.2020.2983962.

[16] Wang, W.—Wang, C.—Wang, Z.—Yuan, M.—Luo, X.—Kurths, J.—Gao, Y.: Abnormal Detection Technology of Industrial Control System Based on Transfer Learning. Applied Mathematics and Computation, Vol. 412, 2022, Art. No. 126539, doi: 10.1016/j.amc.2021.126539.

[17] Street, W. N.—Kim, Y. S.: A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01), 2001, pp. 377–382, doi: 10.1145/502512.502568.

[18] Hulten, G.—Spencer, L.—Domingos, P.: Mining Time-Changing Data Streams. Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01), 2001, pp. 97–106, doi: 10.1145/502512.502529.

[19] Agrawal, R.—Imielinski, T.—Swami, A.: Database Mining: A Performance Perspective. IEEE Transactions on Knowledge and Data Engineering, Vol. 5, 1993, No. 6, pp. 914–925, doi: 10.1109/69.250074.

[20] Barddal, J. P.—Loezer, L.—Enembreck, F.—Lanzuolo, R.: Lessons Learned from Data Stream Classification Applied to Credit Scoring. Expert Systems with Applications, Vol. 162, 2020, Art. No. 113899, doi: 10.1016/j.eswa.2020.113899.

[21] Polikar, R.—Upda, L.—Upda, S. S.—Honavar, V.: Learn++: An Incremental Learning Algorithm for Supervised Neural Networks. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 31, 2001, No. 4, pp. 497–508, doi: 10.1109/5326.983933.

[22] Chen, S.—He, H.: Towards Incremental Learning of Nonstationary Imbalanced Data Stream: A Multiple Selectively Recursive Approach. Evolving Systems, Vol. 2, 2011, No. 1, pp. 35–50, doi: 10.1007/s12530-010-9021-y.

[23] Lu, Y.—Cheung, Y. M.—Tang, Y. Y.: Dynamic Weighted Majority for Incremental Learning of Imbalanced Data Streams with Concept Drift. Proceedings of

the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17), 2017, pp. 2393–2399, doi: 10.24963/ijcai.2017/333.

Haoran Gao received his B.Sc. in computer science and technology from the Shanghai Normal University, Shanghai, China, in 2020. He is currently pursuing his M.Sc. degree with the Department of Computer Science and Technology, Tongji University, Shanghai, China. His current research interests include machine learning and credit scoring.

Zhijun Ding received his Ph.D. degree from Tongji University, Shanghai, China, in 2007. Currently he is Professor with the Department of Computer Science and Technology, Tongji University, Shanghai, China. His research interests include formal engineering, Petri nets, services computing, and mobile internet. He has published more than 100 papers in domestic and international academic journals and conference proceedings.

Meiqin Pan received her Ph.D. degree from the Shandong University of Science and Technology, Qingdao, China, in 2008. Now she is Associate Professor of the School of Business and Management, Shanghai International Studies University. Her research interests are in information systems, data mining and technology, optimization methods. She has published more than 20 papers in domestic and international academic journals and conference proceedings.