Chapter 5
Data Stream Analysis

Data stream is a typical big data. Data stream can be founded in many real-life applications, such as wireless sensor networks, power consumption, information security and financial market. Data stream classification has drawn increasing attention from the data mining community in recent years. Data stream classification in such real-world applications is typically subject to three major challenges: concept drifting, large volumes, and partial labeling. As a result, training examples in data streams can be very diverse and it is very hard to learn accurate models with efficiency. This chapter provides two related research findings in the field. Section 5.1 describes a novel framework for application-driven classification of data streams [1]. The section first reviews the concepts of data stream, then categorizes diverse training examples into four types and assign learning priorities to them. Following the discussion, it derives four learning cases based on the proportion and priority of the different types of training examples. Finally, the respective support vector machine models are presented. Section 5.2 studies the problem of learning from concept drifting data streams with noise, where samples in a data stream may be mislabeled or contain erroneous values [2]. It has three subsections. The first one is about noisy description for data stream, the second one is the ensemble frameworks for mining data stream and the third one is the theoretical studies of the Aggregate Ensemble.

5.1 Application-Driven Classification of Data Streams

5.1.1 Data Streams in Big Data

Recent advances in computing technology and networking architectures have enabled generation and collection of the unprecedented amount of *data streams* of various kinds, such as network traffic data, wireless sensor readings, Web page

Y. Shi, *Advances in Big Data Analytics*,
https://doi.org/10.1007/978-981-16-3607-3_5

visits, online financial transactions and phone call record [3]. Consequently, data stream mining has emerged to be one of the most important research frontiers in data mining. Common stream mining tasks include classification [4, 5], clustering [6] and frequent pattern mining [7]. Among them, data stream classification has drawn particular attention due to its vast real-world applications.

Example 5.1 In wireless sensor networks, data stream classification has been used to monitor environment changes. For example, in the sensor data collected by the Intel Berkeley Research Lab [8], each sensor reading contains information (temperature, humidity, light and sensor voltage) collected from 54 sensors deployed in the lab. The whole stream contains consecutive information recorded over a 2-month period (1 reading per 1–3 min). By using the sensor ID as class label, the learning task is to correctly identify the sensor ID (1 out of 54 sensors) purely based on the sensor data and the corresponding recording time.

Example 5.2 In power consumption analysis, data stream classification has been used to measure power consumptions. For example, the power supply stream collected by an Italian electricity company [8] contains hourly power supply of the company recording the power from two sources: power supplied from main grid and power transformed from other grids. The stream contains 3-year power supply records from 1995 to 1998, and the learning task is to predict which hour (1 out of 24 h) the current power supply belongs to.

Example 5.3 In information security, data stream classification has been widely used to monitor Web traffic streams. For example, the KDDCUP'99 intrusion detection dataset [9] was provided by the MIT Lincoln Labs collecting 9 weeks of raw TCP dump data for a local area network. The learning task is to build a predictive model capable of distinguishing between normal connections and intrusive connections such as DOS (denial-of-service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to local super user privileges), and Probing (surveillance and other probing) attacks.

In these applications, the essential goal is *to efficiently build classification models from data streams for accurate prediction*. Comparing to traditional stationary data, building prediction models from stream data faces three additional challenges:

- Concept drifting. In data streams, hidden patterns continuously change with time [29]. For example, in the wireless sensor stream, lighting during working hours is generally stronger than off-hours. Figure 5.1 illustrates the concept drifting problem, where the classification boundary (concept) continuously drifts from b_1 to b_2, and finally to b_3 down the streams.
- Large volumes. Stream data come rapidly and continuously in large volumes. For example, the wireless sensor stream contains 2,219,803 examples recorded over a 2-month period (1 reading per 1–3 min). It is impossible to maintain all historical stream records for in-depth analysis.
- Partial labeling. Due to large volumes of stream data, it is infeasible to label all stream examples for building classification models. Thus, data streams are

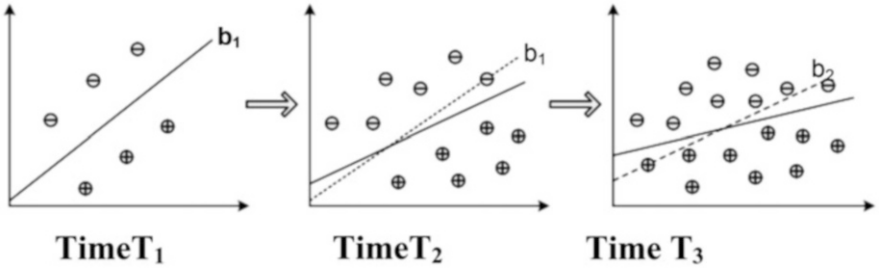

Fig. 5.1 An illustration of concept drifting in data streams. In the three consecutive time stamps T_1, T_2 and T_3, the classification boundary gradually drifts from b_1 to b_2 and finally to b_3

typically partially labeled and training data contain both labeled and unlabeled examples.

As a result, training examples in data streams are very diverse. To see why, let us assume data streams are buffered chunk by chunk. Examples in the most recent *up-to-date chunk* are training data, and examples in the *yet-to-come chunk* are testing data [8]. Due to concept drifting, training examples in the up-to-date chunk often exhibit two distributions: target domain and similar domain, where the former represents the distribution of the testing data, and the latter represents a distribution similar to the target domain [10]. Then, training examples can be categorized into four types: labeled and from the target domain (Type I), labeled and from a similar domain (Type II), unlabeled and from the target domain (Type III) and unlabeled and from a similar domain (Type IV).

In order to build accurate prediction models from such diverse training examples with efficiency, it is necessary to closely examine the characteristics, in particular, the proportion and learning priority, of the different types of examples in the training chunk.

- Proportion. The proportion of training examples from different types is determined by the concept drifting probability and labeling percentage (percentage of labeled examples). For example, when concept drifting is low and labeling percentage is high (low), the raining chunk will have a large portion of Type I (III) examples. When concept drifting is high and labeling percentage is high (low), the training chunk will have a large portion of Type II (IV) examples.
- Learning priority. Generally, examples from the target domains (Types I and III) are capable of capturing the genuine concept of the testing data and have a higher priority than examples from similar domains (Type II and IV). Besides, since Type I examples are labeled, they have a higher priority than Type III examples. Similarly, Type II examples have a higher priority than Type IV examples.

5.1.2 Categorization of Training Examples and Learning Cases

Consider a data stream S consisting of an infinite sequence of examples $\{x_i, y_i\}$, where $x_i \in \mathbb{R}^d$, d is the dimensionality and $y_i \in \{-1, +1\}$ indicates the class label of x_i. Note that y_i may not be always observed. Assume that the stream S arrives at a speed of n examples per second. The decision boundary (concept) underneath drifts with a probability of c, where $0 \leq c \leq 1$. Besides, assume that at each time stamp, a training chunk $D = \{x_1, \cdots, x_n\}$ is buffered and labeled by experts with a labeling rate of 1 per chunk where $0 < l < 1$.

5.1.2.1 Categorization of Training Examples

As discussed previously, due to concept drifting, not all examples in the up-to-date chunk share the same distribution with the testing data in the yet-to-come chunk. In other words, examples in the up-to-date chunk could be generated from some similar domain instead of the target domain. Besides, since it is impractical to label all examples in the up-to-date training chunk, the training chunk will contain both labeled and unlabeled examples. By combining these two factors, we categorize training examples in data streams into four types.

Definition 5.1 (Four types of training examples): In an up-to-date training chunk, there are four types of examples: labeled and from the target domain (Type I), labeled and from a similar domain (Type II), unlabeled and from the target domain (Type III) and unlabeled and from a similar domain (Type IV).

Figure 5.2 illustrates the four types of training examples, where blue solid circles denote the Type I examples, red solid circles denote the Type II examples, blue hollow circles denote the Type III examples, and red hollow circles denote the Type IV examples. Due to the temporal correlation of concepts [11], Type I and Type III examples are usually located at the tail of a training chunk and close to the yet-to-come chunk. Type II and Type IV examples are usually located at the head of a training chunk and relatively far away from the yet-to-come chunk.

Estimation of number of examples By estimating the number of examples of each type, we can gain insights into the training chunk and apply an appropriate learning model. Intuitively, the percentage of labeled examples depends on how fast labeling can be done by the experts, and the number of target domain examples depends on the concept drifting probability. By considering the two factors, the number of examples of each type can be estimated as follows.

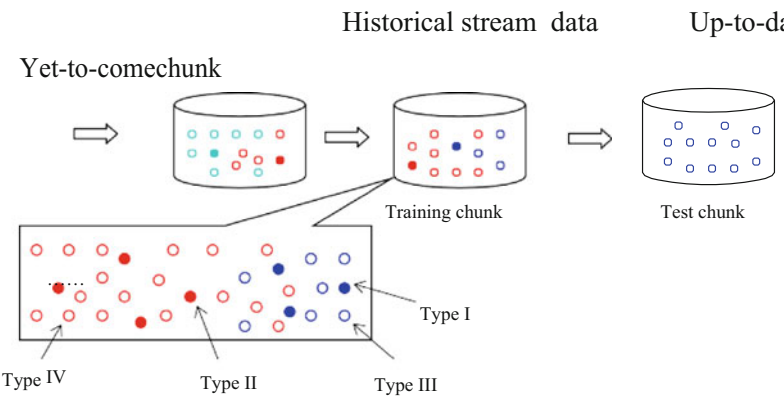

Fig. 5.2 An illustration of the four types of training examples in an up-to-date training chunk

Theorem 5.1 *Let L_1, L_2, L_3 and L_4 be the number of examples of Type I, Type II, Type III and Type IV respectively in the up-to-date chunk. Then,*

$$\begin{aligned} L_1 &\propto \gamma \cdot c^{-1} \cdot l \cdot n \\ L_2 &\propto \left(1 - \gamma \cdot c^{-1}\right) \cdot l \cdot n \\ L_3 &\propto \gamma \cdot c^{-1} \cdot (1 - l) \cdot n \\ L_4 &\propto \left(1 - \gamma \cdot c^{-1}\right) \cdot (1 - l) \cdot n \end{aligned} \tag{5.1}$$

where $\gamma > 0$ is a constant coefficient.

Proof Recall that stream S flows at a speed of n examples per second, the concept drifting probability is c, and the labeling rate is l. The number of target domain examples is inversely proportional to the concept drifting rate c with a coefficient of γ, so it can be easily estimated that $\gamma \cdot c^{-1} \cdot n$ examples in the up-to-date chunk have the same distribution as the testing data. The remaining $(1 - \gamma \cdot c^{-1}) \cdot n$ examples have a similar distribution to the testing examples. From the estimates the theorem follows immediately.

Learning priority Not all the four types of training examples have to be used in model construction. For example, consider a data stream where concept drifting is low and labeling rate is high, the training chunk will have a large portion of Type I examples. In this case, we are able to build a satisfactory model by training only on the Type I examples. We observe that the four types of training examples have the following learning priorities.

Remark 5.1 The learning priority of the four types of training examples are:

$$Type\ I > Type\ III > Type\ II > Type\ IV \tag{5.2}$$

What is the intuition behind Remark 5.1. Generally, examples from the target domain (Type I and Type III) are capable of capturing the genuine concept of the testing data, and thus have a high priority than examples from similar domains (Type II and Type IV). Besides, since Type I examples are labeled, they have a higher priority than Type III examples. Similarly, Type II examples have a higher priority than Type IV examples.

Based on Remark 5.1, when a particular type dominates the training examples, examples with lower priorities will not be used for training. For example, if Type III dominates the training examples, only Type I and Type III examples will be used for training. This is because Type I examples have a higher priority than Type III examples, and the remaining two types have lower priorities. By doing so, we gain in efficiency by building a simple model, comparing to a very complex model if we have to learn from all four types of training examples. On the other hand, the most informative examples are utilized in model construction and the learning accuracy is not sacrificed.

Learning cases Aiming at both accuracy and efficiency in learning prediction models, we categorize learning from data streams into the following four cases.

- Case 1: Type I dominates. When labeling rate is high and concept drifting probability is low, Type I dominates the training examples. In this case, we can train a satisfactory model by using only Type I examples.
- Case 2: Type III dominates. When both labeling rate and concept drifting probability are low, Type III dominates the training examples. According to the learning priority, it is necessary to combine both Type I and Type III examples for training.
- Case 3: Type II dominates. When both labeling rate and concept drifting probability are high, Type II dominates the training examples, and we will use Type I, Type II and Type III examples for training.
- Case 4: Type IV dominates. When labeling rate is low and concept drifting probability is high, Type IV dominates the training examples. This is the most difficult case because most examples are unlabeled and not from the target domain. According to the learning priority, we need to use all the four types of training examples for training.

  These learning cases are further illustrated in Fig. 5.3.

5.1.3 Learning Models of Data Stream

We have introduced the four learning cases. In this section, we present their corresponding learning models.

Throughout the section, $T_1 = (x_1, y_1), \ldots, \left(x_{L_1}, y_{L_1}\right)$ denotes the set of Type I examples. $T_2 = \left\{\left(x_{L_1+1}, y_{L_1+1}\right), \ldots, \left(x_L, y_L\right)\right\}$ denotes the set of Type II exam-

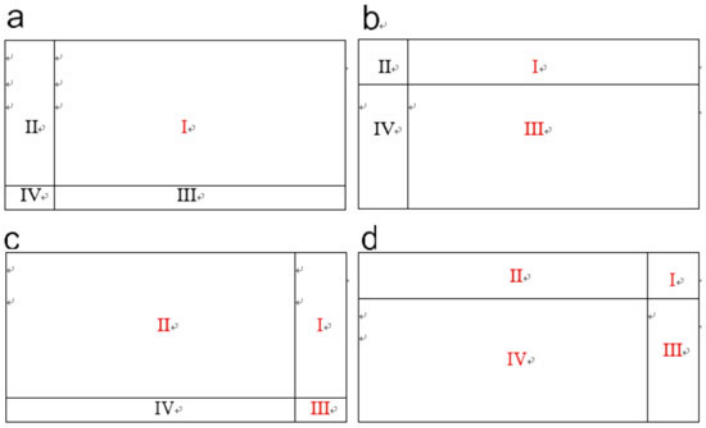

Fig. 5.3 The proportion of the four types of training examples with respect to different labeling rate l and concept drifting probability c. (**a**) l is high and c is low. Case 1, (**b**) both l and c are low. Case 2, (**c**) Both l and l are high. Case 3 and (**d**) l is low and c is high. Case 4

ples, where $L = L_1 + L_2$. $T_3 = \{x_{L+1}, \ldots, x_{L+U}\}$ denotes the set of Type III examples, where U is the set of unlabeled examples. $T_4 = \{x_{L+U+1}, \ldots, x_{L+U+N}\}$ denotes the set of Type IV examples, where N is the set of unlabeled examples.

Case 5.1 Type I Dominates In this case, Type I examples T_1 dominate the training chunk and has the highest learning priority. Thus, only T_1 will be used for training. Formally, to learn from $T_1 = \left\{ (x_1, y_1), \ldots, \left(x_{L_1}, y_{L_1}\right) \right\}$, a generic SVM model can be trained by maximizing the margin distance between classes while minimizing the error rates as,

$$\begin{array}{c} \min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L_1} \xi_i \\ s.t. y_i\,(wx_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0, \quad 1 \leq i \leq L_1 \end{array} \tag{5.3}$$

where w is the projection direction, b is the classification boundary, ξ_i is the error distance from x_i to b, and parameter C is the penalty for the examples inside the margin.

The SVM model given in Eq. (5.3) is a constrained convex optimization problem. To simplify the expression, the Hinge loss function [12] in Fig. 5.4 can be used to transform Eq. (5.3) into an unconstrained convex optimization problem as,

$$\min_{\theta} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L_1} H\left(y_i f_\theta\,(x_i)\right) \tag{5.4}$$

where $\theta = (w, b)$ and $f_\theta(x) = (wx + b)$.

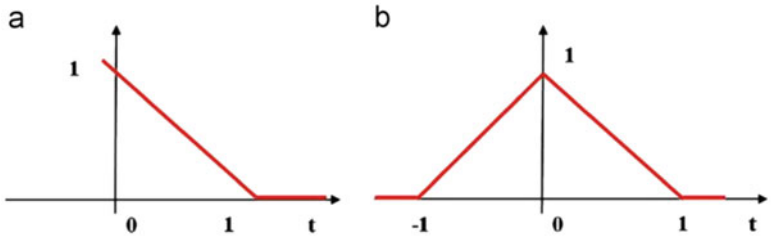

Fig. 5.4 An illustration of the Hinge loss function (**a**) $H(t) = \max(0, 1-t)$, and the Symmetric Hinge loss function (**b**) $H(t) = \max(0, 1-|t|)$. The Hinge loss function is equivalent to the following optimization problem: $\min \xi$ *s. t.* $\xi \geq 0, \xi \geq 1-t$

Case 5.2 Type III Dominates In this case, Type III examples T_3 dominate the training chunk and Type I examples T_1 have a higher learning priority than Type III examples. Thus, both T_1 and T_3 will be used for training.

Learning from T_1 and T_3 is a semi-supervised learning problem [13]. Generally speaking, adding unlabeled T_3 examples into learning will further improve the performance for the following reasons: (1) Labeled examples in T are too few to build a satisfactory model. (2) T_3 contains a relatively large number of examples that come from the target domain, which can greatly help in differentiating the genuine classification boundaries.

Formally, in order to learn from both T_1 and T_3, semi-supervised SVM (TS^{3}VM) [14] can be used as the learning model. The logic behind TS^{3}VM is to find a classification boundary that achieves a maximum margin not only between labeled examples, but also unlabeled examples. That is, adding an extra term $C^* \sum_{i=L+1}^{L+U} H\Big(\mid f_\theta(x_i) \mid \Big)$ to penalize the misclassification of unlabeled examples located inside the margin as,

$$\min_\theta \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L_1} H(y_i f_\theta(x_i)) + C^* \sum_{i=L+1}^{L+U} H(|f_\theta(x_i)|) \tag{5.5}$$

Balance constraint A possible limitation of the TS^{3}VM model is that all unlabeled examples in T_3 may be classified into one class with a very large margin, leading to deteriorated performance. To address this issue, an additional balance constraint should be added to ensure that unlabeled examples in T_3 be assigned into both classes. In the case that we do not have any prior knowledge about the class ratio in T_3, a reasonable approach [12] is to estimate its class ratio from T_1 and T_2 as,

$$\frac{1}{U}\sum_{i=L+1}^{L+U} f_\theta(x_i) = \frac{1}{L_1}\sum_{i=1}^{L_1} y_i \tag{5.6}$$

where L denotes the number of labeled examples and U denotes the number of unlabeled examples.

By taking account of the balance constraint, we can derive a modified semi-supervised SVM model as,

$$\begin{aligned} \min_{\theta} \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{L_1} H\left(y_i f_\theta\left(x_i\right)\right) + C^* \sum_{i=L+1}^{L+U} H\left(\left|f_\theta\left(x_i\right)\right|\right) \\ s.t. \tfrac{1}{U}\sum_{i=L+1}^{L+U} f_\theta\left(x_i\right) = \tfrac{1}{L_1}\sum_{i=1}^{L_1} y_i \end{aligned} \tag{5.7}$$

where $\theta = (w, b)$. Obviously, Eq. (5.7) is a standard TS^{3}VM model and can be easily solved by using off-the-shelf tools [15].

Case 5.3 Type II Dominates In this case, Type II examples T_2 dominate the training chunk, and Type I and Type III examples T_1 and T_3 have higher learning priorities than Type II examples. Thus, T_1, T_2 and T_3 will be used for training.

Accurately learning from these three types of examples is non-trivial. For this purpose, we design a novel transfer semi-supervised SVM model (TS^{3}VM for short). Intuitively, the TS^{3}VM model can be formulated by incorporating examples in T_1, T_2 and T_3 sequentially. Specifically, we can first formulate a generic SVM model by taking T_1 into consideration. Then, a transfer SVM model can be formulated by taking T_2 into consideration.

Finally, we can include T_2 and formulate the TS^{3}VM model.

Learning from T_1 has been discussed in Eq. (5.4), based on which T_2 can be incorporated by applying the transfer learning strategy. Practically, transfer learning can use labeled examples in T_2 to refine the classification boundary by transferring the knowledge from T_2 to T_1. An effective way of doing so is to consider the problem as a multi-task learning procedure [16]. A common two-task learning SVM model on $T_1$and T_2 can be formulated as,

$$\begin{aligned} &\min \tfrac{1}{2}\|w\|^2 + C_1\|v_1\|^2 + C_2\|v_2\|^2 + C\sum_{i=1}^{L} \xi_i \\ &s.t. \quad y_i\left(\left(w + v_1\right)x_i + b\right) \geq 1 - \xi_i, \quad 1 \leq i \leq L_1 \\ &\quad y_i\left(\left(w + v_2\right)x_i + b\right) \geq 1 - \xi_i, \quad L_1 + 1 \leq i \leq L \\ &\quad \xi_i \geq 0, \quad 1 \leq i \leq L \end{aligned} \tag{5.8}$$

where parameters C_1 and C_2 are the penalties on the two tasks, and v_1 and v_2 are the discrepancies between the global optimal decision boundary w and the local optimal decision boundary (i.e., $w + v_1$ for the task of learning from T_1 and $w + v_2$ for the task of learning from T_2).

In Eq. (5.8), parameters C_1 and C_2 control the preference between the two tasks. If $C_1 > C_2$, task 1 is preferred over task 2; otherwise, task 2 is preferred over task 1. By using the Hinge loss function, Eq. (5.8) can be transformed into an unconstrained

form,

$$\min_{\theta} \frac{1}{2}\|w\|^2 + C_1\|V_1\|^2 + C_2\|V_2\|^2 + C\sum_{i=1}^{L} H\left(y_i f_\theta\left(x_i\right)\right) \tag{5.9}$$

where $\theta = (w, v_1, v_2, b), f_\theta(x) = (w + v_1)x + b$ for task 1 and $f_\theta(x) = (w + v_2)x + b$ for task 2.

In addition to T_1 and T_2, an additional semi-supervised learning method can be used to learn from the remaining T_3. As we discussed in Eq. (5.5), by adding an extra term $C^* \sum_{i=L+1}^{L+U} H\left(\left|f_\theta\left(x_i\right)\right|\right)$ to penalize the misclassification of unlabeled examples in T_3 located inside the margin decided by Eq. (5.9), as well as the balance constraint in Eq. (5.6), we can finally get the TS^{3}VM model as,

$$\begin{array}{c} \min_{\theta} \frac{1}{2}\|w\|^2 + C_1\|v_1\|^2 + C_2\|v_2\|^2 \\ + C\sum_{i=1}^{L} H\left(y_i f_\theta\left(x_i\right)\right) + C^*\sum_{i=L+1}^{L+U} H\left(\left|f_\theta\left(x_i\right)\right|\right) \\ s.t. \frac{1}{U}\sum_{i=L+1}^{L+U} f_\theta\left(x_i\right) = \frac{1}{L}\sum_{i=1}^{L} y_i \end{array} \tag{5.10}$$

where $\theta = (w, v_1, v_2, b), f_\theta(x_i) = (w + v_1)x_i + b$ for $1 \le i \le L_1, f_\theta(x_i) = (w + v_2)x_i + b$ for $L_1 + 1 \le i \le L$, and $f_\theta(x_i) = wx_i + b$ for $L + 1 \le i \le L + U$.

5.1.3.1 Solution to the TS^{3}VM Objective Function

As shown in Eq. (5.10), optimizing the objective function of TS^{3}VM is a non-convex optimization problem, which is difficult to find global minima especially for large scale problems. We propose to solve this non-convex problem by using Concave-Convex Procedure (CCCP), which has been developed by the optimization community [6, 10, 26]. CCCP decomposes a non-convex function into the sum of a convex function and a concave function, and then approximates the concave part by using a linear function (a tangential approximation). By doing so, the whole optimization procedure can be carried out iteratively by solving a sequence of convex problems. Algorithm 5.1 describes the CCCP algorithm in detail.

Algorithm 5.1 CCCP Algorithm

Input: the objective function $\mathrm{J}(\theta)$

1: Get the initial point θ_0 with a best guess
2: $\mathrm{J}(\theta) = \mathrm{J}_{\mathrm{vex}}(\theta) + \mathrm{J}_{\mathrm{cav}}(\theta)$
3: **repeat**
4: $\quad \theta_{t+1} = \arg\min_{\theta} \mathrm{J}_{\mathrm{vex}}(\theta) + \mathrm{J}'_{\mathrm{cav}}(\theta_t) \cdot \theta$
5: **until** convergence of θ
6: **return** a local minima solution θ^*

From the CCCP perspective, we can observe that the first four terms of TS^{3}VM are convex functions, whereas the last Symmetric Hinge loss part $C^* \sum_{i=L+1}^{L+U} H\left(|f_\theta(x_i)|\right)$ makes it a non-convex model. Thus, we will decompose and analyze the last part by using the CCCP method. To simplify the notation, we denote $z_i = f_\theta(x_i)$, so the last part can be rewritten as $C^* \sum_{i=L+1}^{L+U} H\Big(\mid z_i \mid \Big)$. Considering a specific z_i (without loss of generality, we denote it as z here), the Symmetric Hinge loss on z can be denoted by $J(z)$ as,

$$J(z) = C^* H\Big(|z|\Big) \tag{5.11}$$

Equation (5.11) is a non-convex function, which can be split into a convex part and a concave part as,

$$J(z) = C^* H\left(|z|\right) = \underbrace{C^* \max\left(0, 1 - |z|\right) + C^* \mid z \mid}_{J_{vex}(t)} \underbrace{- C^* \mid z \mid}_{J_{cav}(t)} \tag{5.12}$$

According to Algorithm 5.1, the next iterative point can be calculated by the approximation of the concave part J_{cav} as,

$$\frac{\partial J_{cav}\Big(z\Big)}{\partial z} \cdot z = \begin{cases} C^* z, & z < 0 \\ -C^* z, & z \geq 0 \end{cases} \tag{5.13}$$

and then minimizing,

$$J(z) = C^* \cdot \max\left(0, 1 - |z|\right) + C^* \mid z \mid + \frac{\partial J_{cav}\Big(z\Big)}{\partial z} z \tag{5.14}$$

If $z < 0$ in the current iteration, then in the next iteration, the current effective loss can be denoted as

$$L\left(z, -1\right) = C^* \max\left(0, 1 - |z|\right) + C^* \mid z \mid + C^* z = \begin{cases} 2C^* z, & z \geq 1 \\ C^*\left(1 + z\right), & \mid z \mid < 1 \\ 0, & z \leq -1 \end{cases} \tag{5.15}$$

On the other hand, if $z > 0$, then in the next iteration, the current effective loss can be denoted as

$$L\left(z, +1\right) = C^* \max\left(0, 1 - |z|\right) + C^* \mid z \mid - C^* z = \begin{cases} 0, & z \geq 1 \\ C^*\left(1 - z\right), & \mid z \mid < 1 \\ -2C^* z, & z \leq -1 \end{cases} \tag{5.16}$$

By doing so, within each iteration, when taking all $z_i = f_\theta(x_i)$ into consideration, solving the TS^{3}VM model is equivalent to solving Eq. (5.17) under the balance constraint Eq. (5.6),

$$\begin{aligned} &\min_{\theta} \frac{1}{2}\|w\|^2 + +C_1\|v_1\|^2 + C_2\|v_2\|^2 \\ &+ C\sum_{i=1}^{L} H\left(y_i f_\theta\left(x_i\right)\right) + \sum_{i=L+1}^{L+U} L\left(f_\theta\left(x_i\right), y_i\right) \end{aligned} \tag{5.17}$$

where $y_i (L + 1 \leq i \leq L + U)$ is the class label of x_i that has been assigned in the previous iteration. If $y_i < 0$, Eq. (5.15) will be used to calculate the loss function; otherwise, Eq. (5.16) will be used to calculate the loss function.

The detailed description of solving TS^{3}VM is given in Algorithm 5.2.

Algorithm 5.2 TS^{3}VM Learning Model

Input: T_1, T_2 and T_3

Use T_1 and T_2 to build a transfer SVM model as shown in Eq. (6.8), and get the initial point $\theta_0 = (w_0, v_{10}, v_{20}, b_0)$

repeat

$y_i \leftarrow \text{sgn}(wx_i + b), \forall L + 1 \leq i \leq L + U$

$\theta \leftarrow$ Calculate Eq. (5.17) under the balance constraint Eq. (5.6)

until y_i remains unchanged, $\forall L + 1 \leq i \leq L + U$

return $f(x) = \text{sgn}(wx + b)$

Theorem 5.2 (Convergence of TS^{3}VM) *The TS^{3}VM learning model in Algorithm 5.2 converges after a limited number of iterations.*

Proof In Algorithm 5.2, in each iteration t, the objective function $J(\theta_t)$ is split into a convex part $J_{vex}(\theta_t)$ and a concave part $J_{cav}(\theta_t)$. Then, in the next iteration $t + 1$, the point θ_{t+1} is the minimal solution of the current objective function, and we have

$$J_{vex}\left(\theta_{t+1}\right) + J'_{cav}\left(\theta_t\right)\theta_{t+1} \leq J_{vex}\left(\theta_t\right) + J'_{cav}\left(\theta_t\right)\theta_t \tag{5.18}$$

Meanwhile, because the concavity of $J_{cav}(\theta)$, we have,

$$J_{cav}\left(\theta_{t+1}\right) \leq J_{cav}\left(\theta_t\right) + J'_{cav}\left(\theta_t\right)\left(\theta_{t+1} - \theta_t\right) \tag{5.19}$$

By adding both sides of Eq. (5.18) and Eq. (5.19), we have

$$\begin{aligned} &J_{vex}\left(\theta_{t+1}\right) + J_{cav}\left(\theta_{t+1}\right) + J'_{cav}\left(\theta_t\right)\theta_{t+1} \\ &\leq J_{vex}\left(\theta_t\right) + J'_{cav}\left(\theta_t\right)\theta_t + J_{cav}\left(\theta_t\right) + J'_{cav}\left(\theta_t\right)\left(\theta_{t+1} - \theta_t\right) \end{aligned} \tag{5.20}$$

Move the third item on the left-hand side of Eq. (5.20) to the right-hand side, we have

$$\begin{aligned} &J_{vex}\left(\theta_{t+1}\right)+J_{cav}\left(\theta_{t+1}\right) \leq J_{vex}\left(\theta_t\right)+J'_{cav}\left(\theta_t\right)\theta_t \\ &+J_{cav}\left(\theta_t\right)+J'_{cav}\left(\theta_t\right)\left(\theta_{t+1}-\theta_t\right)-J'_{cav}\left(\theta_{t+1}\right)\theta_{t+1} \end{aligned} \tag{5.21}$$

The right-hand side of the above inequation equals to $J_{vex}((\theta_t))+J_{cav}(\theta_t)$. Therefore, the objective function will decrease after each iteration $J_{vex}(\theta_{t+1}) \leq J(\theta_t)$.

Consequently, Algorithm 5.2 will converge after a limited number of iterations. In fact, as long as the initial point is carefully selected (i.e., using a multi-task SVM model built on T_1, and T_2 as the initial point), Algorithm 5.2 will converge very fast.

Case 5.4 Type IV Dominates This is the most complex learning case. In this case, Type IV examples T_4 dominate the training chunk and has the lowest learning priority. Thus, it is necessary to use all T_1, T_2, T_3 and T_4 for training.

To solve this learning problem, we design a novel Relational K-means-based Transfer Semi-Supervised learning model (RK-TS^{3}VM for short). The TS^{3}VM model, as discussed previously, is used to learn from T_1, T_2 and T_3. Now we discuss how to learn from T_4 using a Relational K-means model [17] (RK for short).

Learning from T_4 is more challenging than from other three types of training examples, mainly because examples in T_4 are unlabeled and have different distributions from the target domain. The aim of the RK model is to *transfer knowledge from T_4 to T_1, T_2 and T_3 by constructing some new features for the three types of examples using the relational information between T_1, T_2, T_3 and T_4.*

An example of RK learning is shown in Fig. 5.5, where T_4 examples are first clustered into k clusters, $G_1, \cdots, G_k$ based on a relational matrix built between T_1 and T_4. After that, k new features $f(x_i, G_\tau)$ $(\tau = 1 \cdots, k)$ are added to each example x_i in T_1 to construct a new data set T'_1 by calculating the relationship between x_i and each cluster center. By doing so, the new data set T'_1 will contain information transferred from T_4, which can help build a more accurate prediction model.

Given L_1 examples in T_1 and N examples in T_4, the purpose of the relational k-means clustering is to cluster instances in T_4 into k groups, by taking the relationships between instances in T_1 and T_4 into consideration. Let $W \in \mathbb{R}^{L_1 \times N}$ denote the similarity matrix between T_1 and T_4 with each $w_{i,j}$ indicating the

Information from T_1 | Information from T_4 | Class label for T_1

A_1	A_2	..	A_d	G_1	..	G_k	Y
1	2	..	5	$f(x_1, G_1)$	..	$f(x_1, G_k)$	1
..	..	..	..	..	..	..	..
3	7	..	1	$f(x_{L_1}, G_1)$	..	$f(x_{L_1}, G_k)$	2

Fig. 5.5 An illustration of the RK learning model

similarity (which can be calculated according to the Euclidian distance) between instance x_i in T_1 and instance x_j in T_4. For each cluster G_τ on W the average pairwise similarity for all examples in G_τ can be defined as

$$S_{G_\tau} = \frac{1}{|G_\tau|^2} \sum_{x \in G_\tau} \sum_{x' \in G_\tau} S\left(x, x'\right) \tag{5.22}$$

where $S(x, x^{'})$ denotes the similarity between two examples of x and $x^{'}$. On the other hand, the variance of the relationship values of all examples in G_τ can be calculated as

$$\delta_{G_\tau} = \frac{1}{|G_\tau|} \sum_{y_i \in G_\tau} \left(\beta_j - \beta_{G_\tau}\right)^T \left(\beta_j - \beta_{G_\tau}\right) \tag{5.23}$$

where β_{G_τ} denotes the average relationship vector of all instances in G_τ, and $\beta_i \in \mathbb{R}^{1 \times L_1}$ denotes the relationships of instance x_j with respect to all examples in T_1.

The objective of the relational k-means is to find k groups, G_τ, $\tau = 1, \cdots, k$, such that the sum of the similarities is maximized while the sum of variances is minimized as

$$J_e^{'} = \max \sum_{\tau=1}^{k} J_{G_\tau} = \max \sum_{\tau=1}^{k} \frac{S_{G_\tau}}{\delta_{G_\tau}} \tag{5.24}$$

Explicitly solving Eq. (5.24) is very difficult. Alternatively, we can use a recursive hill-climbing search process as an approximation solution. Assume that examples in T_4 are clustered into k clusters, $G_1, \cdots, G_k$. Moving an instance x from cluster G_i to cluster G_j changes only the cluster objective values J_{G_i} and J_{G_j}. Therefore, in order to maximize Eq. (5.24), at each step t, we randomly select an example x from a cluster G_i, and move it to cluster G_j. Such a move is accepted only if the Inequity (5.25) achieves a higher value at step $t + 1$.

$$J_{G_i}(t) + J_{G_j}(t) < J_{G_i}(t+1) + J_{G_j}\left(t+1\right) \tag{5.25}$$

Based on the search process in Inequity (5.25), major steps of the relational k-means are listed in Algorithm 5.3.

Algorithm 5.3 has three tiers of loops. Within each tier, it needs to frequently recalculate $J_{G_c}\left(t\right)$ when the current examples are removed from its current group to another. Nevertheless, because $J_{G_c}\left(t\right)$, as shown in Eq. (5.24), contains information from both the similarity S_{G_i} and variance δ_{G_i} in the relationship matrix, frequently recalculating $J_{G_c}\left(t\right)$ will be time-consuming. To alleviate this

problem, we introduce an addictive update method and a subtractive update method to recalculate $J_{G_c}\left(t\right)$.

Algorithm 5.3 Relational k-Means Clustering

Input: T_1, T_4, number of clusters k, and number of iterations T

```
W ← calculate similarity matrix between T1 and T4
G1, ..., Gk ← apply k-means to W
for t ← 1 to T do
    x ← randomly select an example from T4
    Gi ← current cluster of example x
    JGi (t) ← calculate Gi's objective value in Eq. (5.24)
    JGi (t + 1) ← Gi's new value after excluding x
    for j ← 1 to k, j ≠ i do
        JGj (t) ← calculate Gj's objective value
        JGj (t + 1) ← Gj's new value after including x
        if inequity (6.25) is true then
            Gj ← Gj ∪ x; Gi ← Gi \ x
            break
        end if
    end for
end for
μ1, ..., μk ← calculate cluster centers for G1, ..., Gk
return μ1, ..., μk
```

Consider an example x in T_4 that moves from group G_i to G_j. Before the move, β_{G_i} and β_{G_j} are the mean vectors, $\delta_{G_i}$and δ_{G_j} are the variance vectors. After the move, the new groups are G_i' and G_j'. Then the addictive update is given in the following theorem:

Theorem 5.3 (Additive Update) *When adding an example x into G_j, the mean vector of G_j, β_{G_i}, can be updated to β'_{G_i} as follows,*

$$\beta'_{G_j} = \frac{1}{\mid G'_j \mid} \sum_{x_i \in} \beta_l = \frac{1}{n_j + 1}\left(n_j \cdot \beta_{G_j} + \beta_k\right) = \beta_{G_i} + \frac{\beta_k - \beta_{G_j}}{n_j + 1}, \tag{5.26}$$

Meanwhile, the variance δ_{G_j} can be updated to δ'_{G_j} as follows,

$$\begin{aligned} \delta'_{G_j} &= \frac{1}{\left|G'_j\right|} \sum_{y_l \in G'_j} \left(\beta_l - \beta'_{G_j}\right)^T \left(\beta_l - \beta'_{G_j}\right) \\ &= \frac{n_j}{n_j+1}\delta_{G_j} + \frac{n_j}{(n_j+1)^2}\left(\beta_k - \beta_{G_j}\right)^T \left(\beta_k - \beta_{G_j}\right) \end{aligned} \tag{5.27}$$

where n_j is the number of examples in G_j.

Therefore, the updated mean and variance vectors of group G_j can be incrementally calculated, without recalculating Eq. (5.24). Similarly, for a group G_i, where an example x is removed, its mean and variance vectors can be updated using the following theorem.

Theorem 5.4 (Subtractive Update) *When an example x is removed from group G_i, the mean vector β_{G_i} can be updated to β'_{G_i} as follows,*

$$\beta'_{G_i} = \frac{1}{\mid G'_i \mid} \sum_{y_l \in G'_i} \beta_l = \frac{1}{n_i - 1} \left(n_i \cdot \beta_{G_i} - \beta_k \right) = \beta_{G_i} - \frac{\beta_k - \beta_{G_i}}{n_i - 1}, \tag{5.28}$$

Meanwhile, the variance δ_{G_i} can be updated to δ'_{G_i} as follows, where n_i is the number of examples in G_i.

$$\begin{aligned} \delta'_{G_j} &= \frac{1}{|G'_i|} \sum_{y_l \in G'_i} \left(\beta_l - \beta'_{G_i} \right)^T \left(\beta_l - \beta'_{G_i} \right) \\ &= \frac{n_i}{n_i - 1} \delta_{G_i} - \frac{n_i}{(n_i - 1)^2} \left(\beta_k - \beta_{G_i} \right)^T \left(\beta_k - \beta_{G_i} \right) \end{aligned} \tag{5.29}$$

where n_i is the number of examples in G_i.

Time complexity Now we analyze the time complexity of Algorithm 5.3. In Algorithm 5.3, when searching for a new group for each example in the relationship matrix, the updating operation, by using Theorems 5.3 and 5.4, can be executed within constant time $O(1)$. Besides, Algorithm 5.3 is a greedy algorithm. In each iteration, it uses a local optimization technique to cluster examples into groups that maximizes Eq. (5.24). There are three tiers of loops in the algorithm. The first tier aims to find the best group for each example x with the worst-case complexity of $O(k)$ (i.e., traversing all the k groups). The second tier aims to find the best groups for all examples in T_4, which has the worst-case complexity of $O(N)$ (i.e., searching over all the N examples). The last tier aims to make the algorithm converge to a stable solution. Obviously, the first two tiers dominate the time consumption of the whole algorithm, and thus the time complexity of Algorithm 5.3 is $O(k) \times O(N) = O(kN)$.

RK- TS^{3}VM learning model Algorithm 5.4 lists the detailed procedures of the RK- TS^{3}VM learning model, which is the combination of the TS^{3}VM and RK learning models. Given a training chunk D, Step 1 identifies the four types of examples T_1, T_2, T_3 and T_4. Step 2 constructs a group of k feature vectors, denoted by $\mu = \{\mu_1, \cdots, \mu_k\}$, by applying RK to T_1 and T_4. In Step 3 and Step 4, the k new features are appended to each example in T_1, T_2 and T_3 to form three new sets denoted by T'_1, $T'_2$and T'_3, respectively. Step 5 builds a TS^{3}VM model F from T'_1, T'_2, and T'_3. In Step 6, the feature vectors μ and F are combined to form the final prediction model. For any example x in the testing chunk, RK- TS^{3}VM first calculates k new features for x, then uses the TS^{3}VM model to predict a label for x.

Algorithm 5.4 RK-TS^{3}VM Learning Framework

Input: training chunk D, chunk size n, labeled rate l, concept drifting rate c, number of clusters k

Step 1: Identify the four types of data T_1, T_2, T_3, T_4 in D according to the labeled rate l and concept drifting probability c using Eq. (5.1)

Step 2: Using RK model on T_1 and T_4 to get k cluster centers denoted by $\mu = \{\mu_1, \ldots, \mu_k\}$

Step 3: for each instance x in T_1, T_2, and T_3, add k attributes using the inner produce between x and μ

Step 4: Get the new samples T'_1, T'_2, and T'_3 from **Step 3**

Step 5: Construct a TS^3 VM model using T'_1, T'_2, and T'_3, and get the model F

return μ and **F** together as the prediction model

The data analysis of implementing the above algorithms can be found in Zhang et al. [13].

5.2 Robust Ensemble Learning for Mining Noisy Data Streams

5.2.1 Noisy Description for Data Stream

Based on the characteristics of the stream data, existing work roughly describes data streams into the following two styles: stationary data streams [11, 18–20] and dynamic data streams [5, 21–23].

According to the stationary description, if data streams are divided into data chunks as shown in Fig. 5.6, then training data chunks (which include both historical data chunks and the up-to-date chunk) will have a similar or identical distribution to the yet-to-come data chunk. Thus, classifiers built from the training data chunks will have reasonably good performance in classifying data from the yet-to-come data chunk. The advantage of the stationary description is that we may directly apply traditional classification techniques to the data streams. For example, since the up-to-date data chunks have the same distribution as the yet-to-come data chunk, we can collect all historical classifiers to build a classifier ensemble. However, this stationary description takes no consideration of the concept drifting in stream data, so it can hardly, if not impossible, be used to describe most real-world data streams.

Noticing the limitations of the stationary description, a recent work [21] describes the data streams in a dynamic scenario where training chunks have different distributions $p(x,y)$ (where x denotes the feature vector and y denotes the class label) from that of the yet-to-come data chunk, and classifiers built on the training set may perform only slightly better than random guessing or simply predicting all examples to belong to one single class. Comparing to the stationary description, the dynamic description emphasizes on the situation that training data

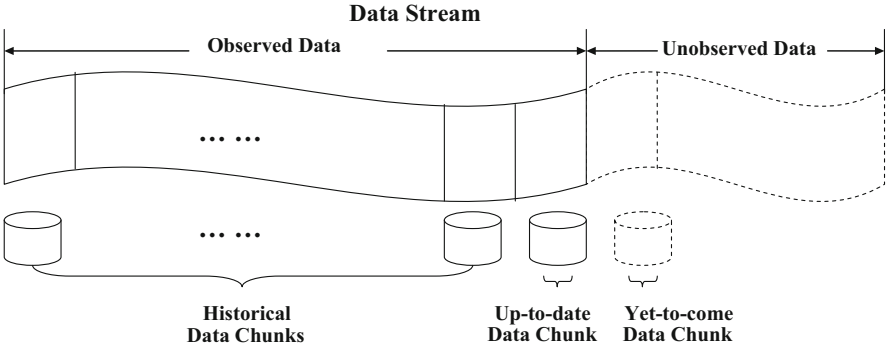

Fig. 5.6 An illustration of the "historical", "up-to-date" and "yet-to-come" data chunks. A data stream can be split into two parts: the observed data stream (which is denoted by the solid lines) and the unobserved data stream (which is denoted by the dotted lines). Assume the data stream is processed chunk-by-chunk. The observed data stream can be further categorized into two types: the latest data chunk is called the "up-to-date" chunk, while the remaining data chunks are called the "historical" data chunks. Besides, the "yet-to-come" data chunk is the first data chunk of the unobserved data streams

chunks do not necessarily have the same distribution as the yet-to-come data chunk. Under this description, building classifiers from the up-to-date data chunk to predict the yet-to-come data chunk is better than building classifiers from the aggregation of all historical chunks because the buffered chunks (probably outdated with respect to the newly arrived data chunk) will deteriorate the ensemble performance. In a narrow sense, this dynamic description is much looser than the stationary description, which makes it more applicable for mining concept drifting data streams. However, the disadvantage of the dynamic description is also obvious, in the sense that it doesn't discriminate concept drifting from data errors. If the up-to-date data chunk contains noisy samples, building classifiers on this noisy data chunk to predict the yet-to-come data chunk may cause more errors than using a classifier ensemble built on previously buffered data chunks. Consequently, although the dynamic description is more reasonable than the stationary description for data streams, in practice, it is still not capable of describing all the realistic data streams.

Consider a data stream management system whose buffer contains five consecutive data chunks as shown in Fig. 5.7. The stationary description can only cover the process from D_1 to D_2, where the distribution $p_1(x,y)$ remains unchanged. The dynamic description covers the process from D_2 to D_3, where the concept drifts from $p_1(x,y)$ to $p_2(x,y)$ without being interrupted by noisy data chunks. A more general situation, as depicted in the process from D_3 to D_5, is that the concept drifting ($p_2(x,y)$ evolves to $p_3(x,y)$) is mixed with noise (a noisy data chunk D_4 is observed). To explicitly describe this type of data streams, we define a noisy description of data streams as follows:

Noisy Description for Data streams Mining from real-world data streams may confront the challenges of concept drifting and data errors simultaneously.

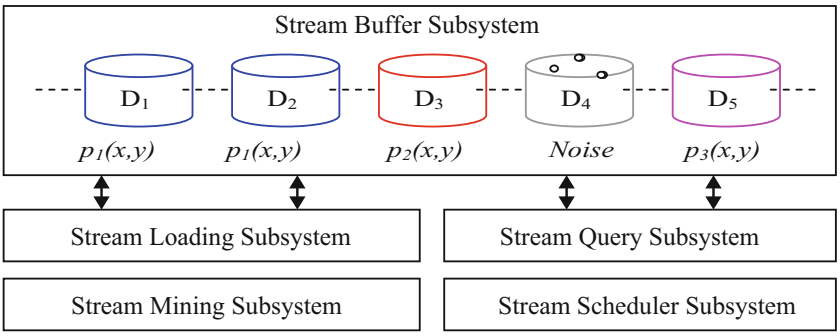

Fig. 5.7 A conceptual view of noisy data in data stream management system. The data stream management system can be separated into five parts: a stream buffer subsystem, a stream loading subsystem, a stream query subsystem, a stream mining subsystem, and a stream scheduler subsystem. In the stream buffer subsystem, there are five buffered data chunks, $D_1, D_2, \ldots, D_5$, of which D_4 is a noise data chunk. D_1 and D_2 share the same distribution $P_1(x,y)$. From D_2 to D_3, the underlying concept changes from $P_1(x,y)$ to $P_2(x,y)$. From D_3 to D_4 and finally to D_5, the concept changes from $P_2(x,y)$ to $P_3(x,y)$, meanwhile, a noisy chunk D_4 is observed between D_3 and D_5. The stationary description of data streams can only cover the process from D_1 to D_2, while the dynamic description of data streams only covers the process from D_2 to D_3. Our noisy description covers a much more common process from D_3 to D_5

The noisy description addresses both concept drifting and data errors in a data stream management system. It is much more general than the stationary and dynamic descriptions. It then can be adapted for generic data streams.

5.2.2 *Ensemble Frameworks for Mining Data Stream*

The nature of continuous volumes of the stream data raises the needs of designing effective classifiers with high accuracy in predicting future testing instances as well as good efficiency in handling massive volumes of training instances. In the past few years, many solutions have been proposed to build prediction models from data streams. An early solution is to build model by using online incremental methods [18, 19] which update a single model by incorporating newly arrived data. During the learning process, incremental methods continuously revise the model to discover new patterns in the most recent data chunk. For example, Domingos and Hulten [10] introduced an ultra fast decision tree learner VFDT which incrementally builds Hoeffding trees from the high-volume data streams. Similar approach was extended to CVFDT [19] which handles time changing and concept drifting streams. By doing so, most of the incremental methods violate the efficiency rule because updating a classifier according to the newly arrived data can be a time-consuming process. An alternative solution is to build a single and simple classifier on the up-to-date chunk without considering historical data chunks, i.e., discarding old classifiers and

rebuilding a new classifier on the new data chunk. This build-then-discard method, unfortunately, may not work well because of the important loss incurred by the discarded classifiers. To overcome this challenge, a number of ensemble methods have been proposed.

Different from the incremental learning where the goal is to deliver a single model, ensemble learning intends to produce a number of models and relies on their voting for final predictions. Such design brings two advantages for ensemble learning to handle data streams: (1) because models are trained from a small portion of stream data, it can efficiently handle streams with fast growing data volumes; and (2) because the final predictions are the voting of a number of base models, the concept drifting in the stream can be adaptively and rapidly addressed by changing the weight value of each voting member. For example, Street and Kim [24] proposed a SEA algorithm, which combines decision tree models using majority-voting. Kolter and Maloof [25] proposed an ensemble method by using weighted online learners to handle drifting concepts. Wang et al. [11] proposed a weighted ensemble, in which they assign each classifier a weight reversely proportional to the classifier's accuracy on the most recent data chunk. Yang et al. [26] proposed proactive learning where concepts (models) learnt from previous chunks are used to foresee the best model to predict data in the current chunk. Zhu et al. [27] proposed an active learning framework to selectively label instances for concept drifting data streams. Gao et al. [21] proposed to build different base classifiers on a most recent data chunk to construct the classifier ensemble.

In summary, the above ensemble frameworks for stream data mining can be roughly categorized into the following two categories, according to their ways of forming the base classifiers: horizontal ensemble (including weighted ensemble) frameworks which build base classifiers using several buffered data chunks (as illustrated in Fig. 5.8a), and vertical ensemble framework which build base classifiers on the up-to-date data chunk using different algorithms (as illustrated in Fig. 5.8b).

5.2.2.1 Horizontal Ensemble and Weighted Ensemble Frameworks

Consider a data stream containing an infinite number of data chunks $\{Di\}_{i=1}^{+\infty}$. Due to the limitation of the storage space, the system buffer can only accommodate at most n consecutive chunks each of which contains a certain number of instances. Assume at the current time stamp we are observing the n^{th} chunk D_n, and the buffered data chunks are denoted by $D_1, D_2, \ldots, D_n$. In order to predict data in a newly arrived chunk D_{n+1}, one can choose a learning algorithm L to build a base classifier f_i from each of the buffered data chunks D_i, say $f_i = \mathcal{L}(D_i)$, and then predict each instance x in D_{n+1} by combining the predictions of the base classifiers f_i $(i = 1, 2, \ldots, N)$ to form a classifier ensemble through the model averaging mechanism shown in Eq. (5.30) [11, 25, 27, 28]:

$$f_{HE}(x) = \frac{1}{N}\sum\nolimits_{i=1}^{N} f_i(x) \tag{5.30}$$

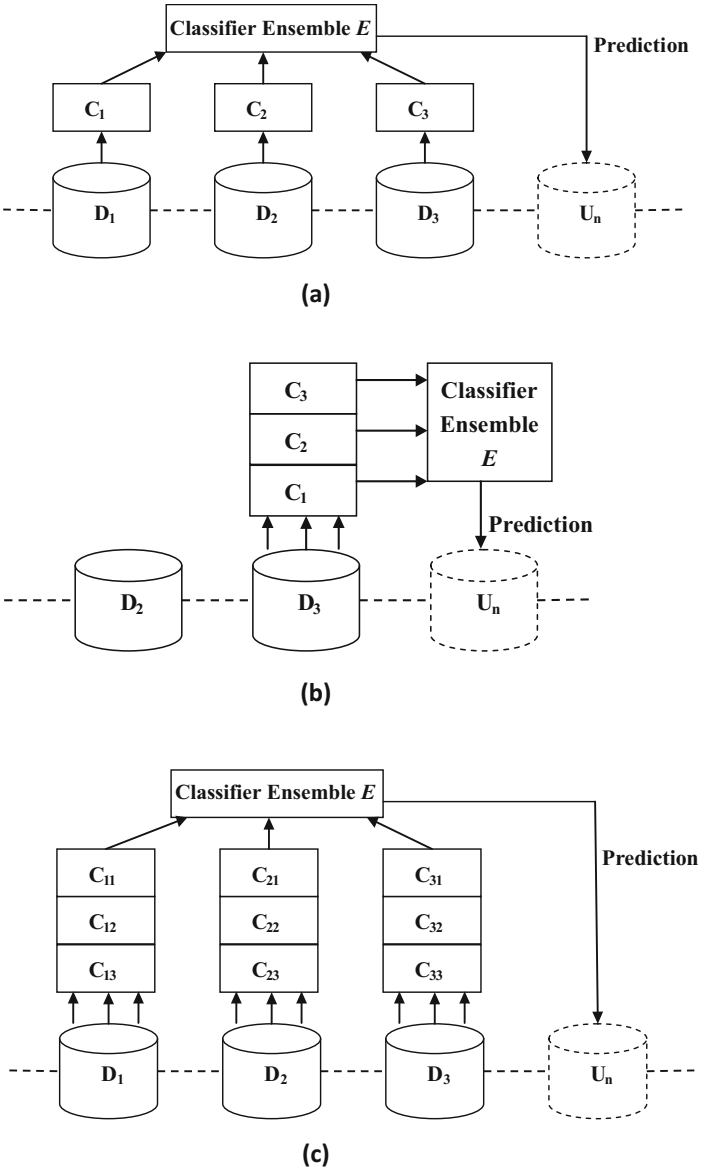

Fig. 5.8 A conceptual flowchart of the classifier ensemble framework for stream data mining where (**a**) shows the horizontal ensemble framework, which builds different classifiers on different data chunks; (**b**) shows the vertical ensemble framework, which builds different classifiers on the up-to-date data chunk with different learning algorithms; and (**c**) shows the aggregate ensemble framework, which builds classifiers on different data chunks using different learning algorithms

An alternative version of the horizontal ensemble is to add weight values to the base classifiers [11, 27]. Different from the model averaging, a weighted ensemble minimizes the variance error e_v of each base classifier on the up-to-date data chunk, then assigns each classifier a weight that is reversely proportional to the error rate e_v. The advantage of the horizontal ensemble and weighted ensemble is twofold: (1) they can reuse information of the buffered data chunks, which may be beneficial for the testing data chunk; and (2) they are robust to noisy streams because the final decisions are based on the classifiers trained from different chunks. Even if noisy data chunks may deteriorate some base classifiers, the ensemble can still maintain relatively stable prediction accuracy. The disadvantage of such an ensemble framework, however, lies in the fact that if the concepts of the stream continuously change, information contained in previously buffered classifiers may be invalid to the current data chunk. As a result, combining old-fashioned classifiers may not improve the overall prediction accuracy. In summary, both horizontal and weighted ensembles, in fact, are based on the stationary description of the data streams that buffered data chunks share similar or identical distributions to the yet-to-come data chunk, such that information in the buffered data chunks can be used to predict the yet-to-come data chunk.

5.2.2.2 Vertical Ensemble Framework

Assume we have m learning algorithms L_j ($j = 1,2,\ldots,m$), a vertical ensemble [17] builds base classifiers using each algorithm on the up-to-date data chunk D_n as $f_j = \mathcal{L}_j(D_n)$, and then combines all base classifiers through model averaging as given in Eq. (5.31),

$$f_{VE}^{n}(x) = \frac{1}{m}\sum_{i=1}^{m} f_{in}(x) \tag{5.31}$$

In the case that prior knowledge of the yet-to-come data chunk is unknown, model averaging on the most recent chunk can achieve minimum expectation error on the test set. In other words, building classifiers using different learning algorithms can decrease the expected bias error compared to any single classifiers. For example, assuming a data stream whose joint probability $p(x,y)$ evolves continuously, if we only use a stable learner such as SVM, then SVM may perform better than an unstable classifier when $p(x)$ changes while $p(y|x)$ remains unchanged. On the other hand, if we only use an unstable learner such as decision trees, then decision trees may perform better than SVM when $p(x)$ does not evolve much but $p(y|x)$ changes dramatically. When we have no prior knowledge on whether the evolving of $p(x,y)$ is triggered by $p(x)$ or $p(y|x)$, it is difficult to determine whether a stable classifier or an unstable classifier is better, so combining these two types of classifiers is likely to be a better solution than simply choosing either of them. Although the vertical ensemble has a much looser condition (distribution $p(x,y)$ may continuously change)

than the stationary description (distribution *p(x,y)* remains unchanged), it also has a severe pitfall for realistic data streams. The vertical ensemble builds classifiers only on a single up-to-date data chunk, but as we have discussed before, a realistic data stream system may contain data errors. If the up-to-date data chunk is a noisy data chunk, the results may suffer from severe performance deterioration. Without realizing the noise problems, the vertical ensemble limits itself merely to the concept drifting scenarios, but not to the realistic data streams.

5.2.2.3 Aggregate Ensemble Framework

The disadvantages of the above two ensemble frameworks motivate the proposed Aggregate Ensemble framework (which is illustrated in Fig. 5.8c). We first use m learning algorithms L_i ($i = 1, 2, \ldots, m$) to build classifiers on n buffered data chunks j ($j = 1, \ldots, n$), and then train m-by-n base classifiers $f_{ij} = \mathcal{L}_i\left(D_j\right)$, where i denotes the i^{th} algorithm, and j denotes the j^{th} data chunk. Then we combine these base classifiers to form an aggregate ensemble through model averaging defined in Eq. (5.32), which indicates that the aggregate ensemble is a mixture of the horizontal ensemble and vertical ensemble, and its base classifiers constitute a *Classifier Matrix* (*CM*) in Eq. (5.33).

$$f_{AE} = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij}(x) \tag{5.32}$$

$$CM = \begin{bmatrix} f_{11} & f_{12} \cdots\cdots f_{1n} \\ f_{21} & f_{22} \cdots\cdots f_{2n} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ f_{m1} & f_{m2} \cdots\cdots f_{mn} \end{bmatrix}_{m*n} \tag{5.33}$$

In Eq. (5.33), each element f_{ij} in *CM* represents a base classifier built by using algorithm i on data chunk j. As we have mentioned in the vertical ensemble, classifiers on each column of *CM* (*i.e.*, classifiers built on the same data chunk using different learning algorithms) are used to reduce the expected classifier bias error on unknown test data. Classifiers on each row of *CM* (*i.e.*, classifiers built on different data chunks using the same learning algorithm) are used to reduce the impact of noisy data chunks. For example, when the up-to-date training chunk is a noisy chunk, combining classifiers built from the historical data chunks may alleviate the noisy impact. By building a classifier matrix *CM*, the aggregate ensemble is capable of solving a realistic data stream containing both concept drifting and data errors.

5.2.3 Theoretical Studies of the Aggregate Ensemble

5.2.3.1 Performance Study of AE Framework

In this section, we explore why and when AE performs better than HE and VE methods. As we have described in the above section, on each data chunk, the aggregate ensemble builds m classifiers by using m different learning algorithms. For a specific test instance x in the yet-to-come data chunk, the horizontal ensemble uses classifiers on a row in matrix CM to predict x, *i.e.*, if we choose learning algorithm i ($1 \leq i \leq m$), then the horizontal ensemble can be denoted by Eq. (5.34)

$$f_{HE}^{i}(x) = \frac{1}{n}\sum_{j=1}^{n} f_{ij}(x) \tag{5.34}$$

The vertical ensemble can be denoted by model averaging on the last column (column n) of the Matrix CM, which is given in Eq. (5.35)

$$f_{VE}^{n}(x) = \frac{1}{m}\sum_{i=1}^{m} f_{in}(x) \tag{5.35}$$

An aggregate ensemble combines all classifiers in CM as base classifiers, through the averaging rule defined by Eq. (5.35). Accordingly, the horizontal ensemble and vertical ensemble are, in fact, two special cases of the aggregate ensemble. Gao et al. [21] proved that in data stream scenario, the performance of a single classifier within a classifier ensemble is expected to be inferior to the performance of the entire classifier ensemble. The horizontal ensemble and vertical ensemble, as special cases of the aggregate ensemble, are not expected as good as the aggregate ensemble. For example, when combining each column in CM, one can have a variant of CM as $CM_c = [g_1, g_2, \ldots, g_n]$, where each $g_i = [f_{1i}, f_{2i}, \ldots, f_{mi}]^T$ is independent of each other and shares the same distribution, say $p(g)$. Then the mean squared error of the horizontal ensemble (with the i^{th} learning algorithm) on a test instance x (with class label y) can be denoted by

$$MSE_{HE}^{i}(x) = E_{p(g)}(y - g_i(x))^2 = y^2 - 2y \cdot E_{p(g)}g_i(x) + E_{p(g)}g_i^2(x) \tag{5.36}$$

For the aggregate ensemble, the mean squared error on x can be calculated as

$$MSE_{AE}(x) = E_{p(g)}\big(y - E_{p(g)}g_i(x)\big)^2 = y^2 - 2y \cdot E_{p(g)}g_i(x) + E_{p(g)}^2 g_i(x) \tag{5.37}$$

Thus, the difference between Eqs. (5.35) and (5.36) is denoted by Eq. (5.38),

$$MSE_{AE}(x) - MSE^{i}_{HE}(x) = E^2_{p(g)} g_i(x) - E_{p(g)} g_i^2(x) \leq 0. \quad \left(\text{since } E^2(x) \leq E\left(x^2\right)\right) \tag{5.38}$$

Accordingly, we assert that the error rate of the aggregate ensemble is expected to be less or equal to the error rate of the horizontal ensemble. Similarly, if we regard *CM* as a column vector where each element is a combination of different rows in *CM*, we can show that the mean squared error of the aggregate ensemble is also expected to be less or equal to that of the vertical ensemble.

In the following we provide some intuitive explanations on why and when AE performs better than HE and VE by using two toy examples in Figs. 5.9 and 5.10. Note that our comparisons here are rather intuitive and qualitative, and rigorous numeric comparisons will be reported in the experimental results in the next section. As shown in Fig. 5.8, assume that AE is trained using three learning algorithms M_1, M_2, and M_3, where $HE(M_i)$ denotes an HE model trained using learning algorithm M_i. For each model, we list three results: (1) training accuracy at time A, (2) test accuracy at time A, and (3) test accuracy at time B which immediately follows A. We can observe that for concept drifting data streams, it is difficult to find a single "optimal" learning algorithm with the best performance across the whole stream. For example, model $HE(M_2)$ has the best prediction accuracy at time stamp A, but unfortunately, it has the worst performance at the next time stamp B. Model $HE(M_3)$ has the worst performance at time A, but it performs the best at time stamp B. On the other hand, AE can guarantee the most reliable performance by combining different learning algorithms. This is because in dynamic data stream

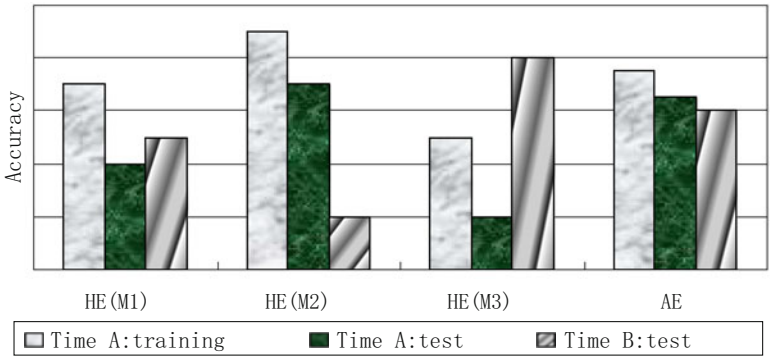

Fig. 5.9 A toy example for comparisons between AE and three HE ensemble methods trained with different learning algorithms (*i.e.*, algorithms M_1, M_2, and M_3). For each ensemble method, three results (bars) are listed for comparisons. The left bar denotes the training accuracy at time A, the bar in the middle denotes the test accuracy at time A, and the bar on the right denotes the test accuracy at time B which follows time stamp A. It is obvious that at time A, the higher the training accuracy, the better the prediction result. However, this result doesn't hold when the concept drifts at the next time stamp B

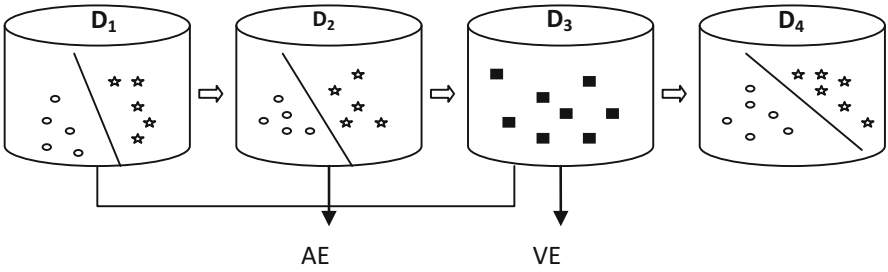

Fig. 5.10 A toy example for comparison between AE and VE. The concept (*i.e.*, the classification boundary) drifts marginally from chunk D_1 to D_2, and finally to D_4. Notice that the up-to-date chunk D_3 is a noisy chunk that carries useless or erroneous. Information when predicting the yet-to-come data chunk D_4

environments it is essentially difficult to know which learning algorithm performs the best at a particular time point. By integrating different learning algorithms as a unified model, we can expect AE to have the smallest variance error and thus have the best prediction accuracy.

AE performs better than VE when the concept drifts marginally and the up-to-date training chunk contains a significant amount of noisy samples. As illustrated in Fig. 5.10, assume that the concept drifts slightly along data chunks, and the up-to-date chunk D_3 is a noisy chunk. VE built on the up-to-date chunk D_3 will show deteriorated performance in predicting D_4. On the other hand, AE can largely avoid such a limitation by incorporating information from classifiers trained from the historical data chunks D_1 and D_2.

Although we have demonstrated that AE, on average, outperforms HE and VE, we are not claiming that AE always performs the best in data stream scenarios. For example, HE may outperform AE if the concept drifts marginally in data streams. In this case, the joint probability distribution $p(x,y)$ will stay stable across the data streams, and thus we can select a strong learning algorithm (*i.e.*, SVM) to construct HE and expect HE to outperform AE. On the other hand, VE may outperform AE if the concept drifts significantly and the up-to-date chunk contains very few noisy samples. In such a case, old-fashioned historical information in AE will deteriorate the learner performance even worse.

5.2.3.2 Time Complexity Analysis

In this section, we study the time complex of the AE framework and discuss whether it is a suitable model, from computational cost perspective, for mining noisy data streams. As discussed above, compared to its peers, AE combines much more base classifiers to build an ensemble predictor. This raises the concern on the efficiency of AE due to its additional cost for training extra base classifiers.

To study AE's time complexity, let's consider the following example. Assume the buffer of the system contains d data chunks, each of which contains N instances. Assume further that m learning algorithms are used to build models. Each time when a new data chunk arrives, we need to follow two steps to update an ensemble: (1) build new base classifier(s) on the new data chunk; and (2) update classifier ensemble by incorporating new base classifier(s). Without loss of generality, we assume that training a new base classifier needs $O(N \lg N)$ time on average, while updating the classifier ensemble to include one base classifier requires $O(\Gamma)$ time, where Γ is related to the dimensionality of attributes. Then the updating of the HE ensemble for each new data chunk needs to (1) build a new base classifier (which costs $O(N \lg N)$time), and then (2) combine the most recent d base classifiers (which costs $O(d\Gamma)$ time) together for prediction. The total time cost can be calculated by Eq. (5.39),

$$O(HE) = O(N \lg N) + O(d\Gamma). \tag{5.39}$$

Since training a base classifier dominates the total cost (*i.e.*, $O(\Gamma) << O(N \lg N)$), and the number of data chunks d in the buffer is rather small. The time complexity of the HE ensemble can be simplified as in Eq. (5.40),

$$O(HE) = O(N \lg N) + O(d\Gamma) = O(N \lg N). \tag{5.40}$$

In comparison, VE builds m base classifiers for each new data chunk. Accordingly, its time complexity $O(VE)$ can be calculated by Eq. (5.41),

$$O(VE) = O(m) * (O(N \lg N) + O(\Gamma)) = O(mN \lg N) \tag{5.41}$$

For AE, it first builds m classifiers when a new data chunk arrives and combines all the $d*m$ base classifiers to build an ensemble. Therefore, its time complexity can be calculated by Eq. (5.42),

$$O(AE) = O(mN \lg N) + O(dm\Gamma) = O(mN \lg N). \tag{5.42}$$

Combining Eqs. (5.40), (5.41), and (5.42), we have the following two conclusions: (1) AE is, asymptotically, as efficient as VE. Both of them have the same time complexity $O(mN \lg N)$; (2) AE requires more time complexity than HE because AE needs to train m base classifiers for each new data chunk. This limitation, in practice, can be alleviated by using a multi-core or multi-processor computing system, where base classifiers can be dispatched and trained on different computation units in parallel. The detailed data analysis of this section can be found in [4].

References

1. Zhang, P., Zhu, X., Shi, Y., Wu, X.: An aggregate ensemble for mining concept drifting data streams with noise. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 1021–1029. Springer, New York (2009)
2. Zhu, X.: Semi-supervised learning literature survey (2005)
3. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 1–16 (2002)
4. Zhang, P., Zhu, X., Shi, Y., Guo, L., Wu, X.: Robust ensemble learning for mining noisy data streams. Decis. Support. Syst. **50**(2), 469–479 (2011)
5. Zhu, X., Zhang, P., Lin, X., Shi, Y.: Active learning from stream data using optimal weight classifier ensemble. IEEE Trans. Syst. Man Cybernetics Part B Cybernetics. **40**(6), 1607–1621 (2010)
6. Aggarwal, C.C., Philip, S.Y., Han, J., Wang, J.: A framework for clustering evolving data streams. In: Proceedings 2003 VLDB Conference, pp. 81–92. Elsevier, Amsterdam (2003)
7. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data Min. Knowl. Disc. **15**(1), 55–86 (2007)
8. Zhang, P., Li, J., Wang, P., Gao, B.J., Zhu, X., Guo, L.: Enabling fast prediction for ensemble models on data streams. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 177–185 (2011)
9. Asuncion, A., Newman, D.: Uci machine learning repository (2007)
10. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80 (2000)
11. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 226–235 (2003)
12. Collobert, R., Sinz, F., Weston, J., Bottou, L., Joachims, T.: Large scale transductive svms. J. Mach. Learn. Res. **7**(8) (2006)
13. Zhang, P., Gao, B.J., Liu, P., Shi, Y., Guo, L.: A framework for application-driven classification of data streams. Neurocomputing. **92**, 170–182 (2012)
14. Chapelle, O., Sindhwani, V., Keerthi, S.S.: Optimization techniques for semi-supervised support vector machines. J. Mach. Learn. Res. **9**(2), 203–233 (2008)
15. Joachims, T., et al.: Transductive inference for text classification using support vector machines. ICML. **99**, 200–209 (1999)
16. Evgeniou, T., Pontil, M.: Regularized multi-task learning. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 109–117 (2004)
17. Zhang, P., Zhu, X., Guo, L.: Mining data streams with labeled and unlabeled training examples. In: 2009 Ninth IEEE International Conference on Data Mining, pp. 627–636. IEEE, New York (2009)
18. Dietterich, T.G.: Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems, pp. 1–15. Springer, New York (2000)
19. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 97–106 (2001)
20. Yuille, A.L., Rangarajan, A.: The concave-convex procedure (cccp). Adv. Neural Inf. Proces. Syst. **2**, 1033–1040 (2002)
21. Gao, J., Fan, W., Han, J.: On appropriate assumptions to mine data streams: analysis and practice. In: Seventh IEEE International Conference on Data Mining (ICDM 2007), pp. 143–152. IEEE, New York (2007)

22. Zhang, P., Zhu, X., Shi, Y.: Categorizing and mining concept drifting data streams. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 812–820 (2008)
23. Zhu, X.: Stream data mining repository. www.cse.fau.edu/xqzhu/S (2009)
24. Street, W.N., Kim, Y.: A streaming ensemble algorithm (sea) for large-scale classification. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377–382 (2001)
25. Kolter, J.Z., Maloof, M.A.: Using additive expert ensembles to cope with concept drift. In: Proceedings of the 22nd International Conference on Machine Learning, pp. 449–456 (2005)
26. Yang, Y., Wu, X., Zhu, X.: Combining proactive and reactive predictions for data streams. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 710–715 (2005)
27. Zhu, X., Jin, R.: Multiple information sources cooperative learning. In: Proceedings of the IJCAI, pp. 1369–1375 (2007)
28. Dai, W., Yang, Q., Xue, G., Yu, Y.: Boosting for transfer learning. In: Proceedings of the ICML, pp. 193–200 (2007)
29. Tsymbal, A.: The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin **106**(2), 58 (2004)