Learning and Detecting Concept Drift

Kyosuke Nishida

A Dissertation

submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Information Science and Technology

in the field of
Synergetic Information Science

at the
Graduate School of Information Science and Technology,
Hokkaido University

February 2008

Abstract

The volume of data that humans create has increased explosively as information science and technology have evolved. Therefore, the demand for learning machines that can extract input-output mappings and knowledge rules from massive data sets has become more urgent, and machine learning is now a core technology in the advanced information society. It has been applied to fields such as pattern recognition, search engines, medical support, robot engineering, image processing, and data mining and has achieved significant accomplishments in each field. Recently, several methods that could not be implemented with older computers have been developed with state-of-the-art computers that have enormous memory capacity and high performance CPUs. Machine learning is expected to continue to develop in the future.

Machine learning can be roughly classified into two types of learning based on how training examples are presented: batch learning and online learning. Batch learning systems are first given a large number of examples that they then learn all at a once. In contrast, online learning systems are given examples sequentially that they learn one by one. Many excellent batch learning systems have been proposed, but there are serious problems with online learning, especially in environments where the statistical properties of the target variable change over time. This change, known as *concept drift*, can happen either gradually or suddenly and significantly. The effectiveness of strategies for building a good learning system depends on the types of changes, so it is difficult to create an ideal learning system. A prime example of concept drift is the spam filtering problem. An effective spam filter must be able to handle various

changes, including changes in the user's criteria for filtering spam, changes in message topics, and changes caused by the crafty people sending spam messages.

There is more and more demand in various fields for a system that can learn sequentially and incrementally and that adapts well to the large amount of data that keeps being created and keeps changing. In this dissertation, taking various approaches to solving the problem of concept drift, we propose online learning systems that are able to learn and detect concept drift. The proposed systems address online classification problems in which they predict a class or category for a sequentially given input.

First, we propose a system called the *a*daptive *c*lassifiers-*e*nsemble (ACE) system that uses multiple classifiers. Using multiple classifiers would be a more natural solution to learning concept drift than continuously revising a single classifier, so several such systems have recently been proposed. The ACE system forms its output by using a weighted majority vote from the outputs of a single online classifier that is continuously updated and many batch classifiers that are not updated. ACE uses confidence intervals for the recent predictive accuracies of batch classifiers to detect concept drift and builds a new batch classifier when concept drift is detected. ACE is able to handle recurring concepts better than conventional systems, but has problems detecting concept drift accurately and is thus too complicated for practical use.

Next, we propose a simpler method of more accurately detecting concept drift. The STEPD method monitors the predictive accuracy of a single online classifier and detects significant decreases in the predictive accuracy, which are caused by concept drift, by using a *s*tatistical *t*est of *e*qual *p*roportions to *d*etect concept drift. When concept drift is detected, the online classifier is reinitialized to prepare for the learning of the next concept. STEPD is able to detect concept drift more quickly and accurately than ACE and other methods, but false alarms degrade the predictive accuracy of STEPD significantly because STEPD uses only a single online classifier.

After STEPD, we propose a system that can reduce the bad influence of false alarms on predictive accuracy, which we call the *t*wo *o*nline classifiers for learning

and *d*etect*i*ng concept drift (Todi) system. When concept drift is detected by the STEPD method, one of the two classifiers is reinitialized, and the other one is not. Using two online classifiers also enables Todi to confirm the correctness of the last detection and to accurately notify a user of the occurrence of concept drift. We evaluated the effectiveness of Todi by using a spam filtering dataset, and the results demonstrate that Todi, with two Bogofilters, performed better than the single Bogofilter (Bogofilter is a well-known and successful spam filters). We plan to combine Todi with ACE to achieve a more intelligent online learning system that can be used to solve various real-world problems.

Finally, we propose a method inspired by human behavior for detecting sudden concept drift, the *l*eaky *i*ntegrate-and-*d*etect (LID) model. We considered the significant difference between humans and machine learning systems, which treat any misclassification as the same misclassification, and studied human change detection in order to apply knowledge from cognitive science to machine learning. We conducted a human behavior experiment to investigate the working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. Results suggested the validity of the working hypothesis, and we noticed that the successive rejection of highly confident classifications is significant for change detection. Using this knowledge, we speculated that the *l*eaky *i*ntegrate-and-*f*ire (LIF) model, which is a spike generation model, could be applied to change detection. We therefore propose the LID model, which is based on the LIF model. LID uses the degree of confidence in and the recent accuracy of a single online classifier to detect sudden and significant changes. Computer experiments show that LID enabled the online classifier to respond to sudden and significant changes quickly and accurately in an environment that also includes noise and gradual changes. We plan to apply knowledge about the degree of confidence to Todi and ACE.

In conclusion, we conducted interdisciplinary studies and propose here online learning systems for learning and detecting concept drift. This dissertation has contributed

to developing the abilities that an online learning system for handling concept drift should have.

Abstract (in Japanese)

情報科学の発展によって人類が創り出すデータ量は爆発的に増大している．このため，大量のデータから入出力関係や知識表現を抽出する機械学習の需要が高まり，現在では高度情報化社会を支える基盤技術の一つにまで成長した．音声・文字認識，検索エンジン，医療支援，ロボット工学，画像処理，データマイニングなど，これまでに機械学習の技術が応用され大きな成果を挙げた分野は多岐に渡る．近年では，計算機の記憶容量や演算処理能力の飛躍的な向上を背景に，過去の計算機では実現不可能だった手法が次々と開発されており，今後も機械学習のさらなる発展が予想される．

機械学習を訓練サンプルの与え方によって大別すると，まとめて与えられた大量のサンプルを一度に学習するバッチ学習と，順次与えられるサンプルを追加的に学習するオンライン学習の二種類に分けられる．このうち，バッチ学習については非常に優れた手法が多数提案されているが，オンライン学習，特に，学習対象の基となる統計的な性質が時間と共に変化する環境下での学習については多くの課題が残されている．この変化は *concept drift* と呼ばれ，緩やかな変化のみならず，突然かつ大きな変化をも含む．変化の性質によって学習システムが採るべき戦略は大きく異なるため，様々な性質の変化を含む問題の解決は非常に難しい． その実例の一つとしてはスパムフィルタリング問題が挙げられる．この問題においてスパムフィルタは，ユーザが持つスパム判定基準の僅かな揺らぎやメールで扱う話題の変化，さらには狡猾なスパム送信者が起こす変化など，日々発生する様々な性質の変化に対応しながら学習を進めなければならない．

今後は様々な分野において，幾多の変遷を経ながら大量に創出され続けるデータを処理するために，逐次的・追加的に学習可能な，そして高い順応性を持ったシステムがより一層必要となるであろう．そこで本論文では，与えられた入力サンプルに対応するクラスを

予測するクラス分類問題全般を対象に，concept drift に対応可能なオンライン学習システムを様々なアプローチから複数提案する．全ての提案システムは concept drift の発生を検出して学習に利用するという特徴を持つ．

まず，我々は，複数のクラス分類器を使用するシステム ACE（*a*daptive *c*lassifiers-*e*nsemble）を提案した．複数のクラス分類器を用いるシステムは，単一のクラス分類器を修正しながら使用するシステムよりも concept drift に対応し易い場合が多く，近年多くの研究が行われている．ACE はシステム全体の出力を，常に更新を行う一つのオンラインクラス分類器と，作成後は更新を行わない多数のバッチクラス分類器の出力の重み付き多数決によって決定する．そして，各クラス分類器の最近のサンプルに対する予測精度とその信頼区間を利用して concept drift の検出を行い，新たなバッチクラス分類器を順次追加していく． ACE は再発する変化に対して従来手法よりも高い性能を実現するが，変化検出の精度とシステム構成の複雑さに問題があった．

次に，concept drift の正確な検出を単純なシステムによって実現することを目指し，新たな変化検出法 STEPD を提案した．STEPD は単一のオンラインクラス分類器についてその予測精度を監視し，concept drift の発生が引き起こす最近の予測精度の急激な悪化を統計的検定（a *s*tatistical *t*est of *e*qual *p*roportions to *d*etect concept drift）によって検出する．そして，検出後はクラス分類器を初期化して新たな学習に備える．STEPD は ACE で使用する手法や他の従来手法に比べ高い検出性能を実現したが，誤検出が発生した場合にシステムの精度が大きく悪化する問題があった．

そこで，誤検出が発生しても予測精度が悪化しない学習システムの実現を目指して，我々は二つのオンラインクラス分類器を利用して学習・検出を行うシステム Todi （*t*wo *o*nline classifiers system for learning and *d*etect*i*ng concept drift）を提案した．Todi は，変化検出後に初期化されるクラス分類器に加えて，検出後も継続して学習を行うクラス分類器を使用する．この工夫によって，Todi は高い予測精度を常に維持し続けると共に，過去の検出の正誤を自ら判断して concept drift の発生をユーザに正しく通知できる様になった．Todi の有効性を検証するためスパムフィルタリング問題に取り組んだところ，高性能なスパムフィルタとして知られる Bogofilter をオンラインクラス分類器として二つ使用した Todi は，単体の Bogofilter よりも高い予測精度を実現した．今後は ACE と

Todi の技術を結合して，多くの実世界問題を解決して行きたい．

最後に，人間が行う変化検出に関する知見を基に，LID（*l*eaky *i*ntegrate-and-*d*etect）手法を提案した．我々は，人間が行う変化検出と，どのような誤分類も全て同様に扱って変化を検出する機械学習手法との間には大きな隔たりがあると考えた．そこで人間の変化検出について調査し，認知科学的知見を機械学習に取り入れることを目指して研究を行った．まず「最近の予測精度が高い状況で確信度の高い回答が否定されるほど，人間は変化を高速に検出できる」という作業仮説を立てて行動実験を行い，仮説の正しさを示唆する結果を得た．そして，自信のある回答が連続して否定されることの重要性を発見した． これらの知見から，スパイク発生機構の単純なモデルである *l*eaky *i*ntegrate-and-*f*ire モデルを変化検出に応用できると考え，これを基にした手法 LID を提案した．LID は単一のオンラインクラス分類器が持つ確信度と最近の予測精度を利用して，突然かつ重大な変化を検出する．その有効性を検証するため計算機実験を行ったところ，ノイズや緩やかな変化が存在する環境下でも，LID は高速かつ正確に変化を検出できた．今後は本研究で得た確信度に関する知見を Todi や ACE に導入していきたい．

以上をまとめると，我々は学際的な研究を行って，concept drift の学習と検出が可能なオンライン学習システムを複数提案した．本論文は concept drift を扱うオンライン学習システムが持つべき能力の実現に貢献を果たした．

Acknowledgments

I first extend my deepest gratitude to my parents and my sister for their unconditional support. I dedicate this dissertation to them.

I express my sincerest gratitude to my adviser, Dr. Koichiro Yamauchi, for all of the help that he has given me in completing this dissertation. I have been fortunate to have an adviser who gave me the freedom to explore on my own. I am deeply grateful to my co-adviser, Dr. Satoru Ishikawa. He has always given me helpful advice. My study of human behavior would never have been possible without him. I am thankful to Prof. Takashi Omori for his warm encouragement. I am also grateful to Prof. Masahito Kurihara, Prof. Masashi Furukawa, Prof. Azuma Ohuchi, and Prof. Mitsuo Wada, for their helpful suggestions and comments in reviewing this dissertation. I would like to acknowledge the Japan Society for the Promotion of Science for its generous financial support.

I would like to thank all of the members of our laboratory. I first extend my deepest gratitude to Kenta Narumi and Shohei Shimada. They are not only students that I have mentored, they are my research partners. I am also grateful to Dr. Akitoshi Ogawa, Dr. Akira Toyomura, Dr. Yu Ohigashi, Mr. Takayuki Ohira, Mr. Kenji Kondo, Mr. Kenji Sato, Mr. Hideyuki Takahashi, Ryuji Oshima, Hibiki Ogura, Yugo Nagata, Genki Saito, Yohei Tadeuchi, Jiro Hayami, Tomohiro Harada, Atsushi Sato, Yuki Togashi, Masato Iwasaki, Masayoshi Sato, Shinpei Masuda. I have enjoyed discussing various topics with them. My special thanks to our clerk, Ms. Ikuko Shiratori, for her support and encouragement. I am also thankful to all of my fellow system administra-

tors in our laboratory, especially *hibiki*, *tadeuchi*, *naruken*, and *justice* who helped me devotedly. We have been the greatest system administrators!

I am grateful to the teachers at the Department of Computer Science and Engineering, Tomakomai National College of Technology, especially Dr. Koichi Matsuda for his kindness. I have been fortunate to get a good education from the college before I entered Hokkaido University. I am also thankful to the teachers at the college for giving me an opportunity to be a part-time lecturer at the college.

I express my sincere gratitude to Mr. Joe Talic for his friendship, English teaching, and careful proof-reading of this dissertation. He has always encouraged me to believe in myself. I would like to keep being a *busy beaver*.

Finally, I send my best regards to all my friends. Without all of you being there, this dissertation would never have been completed.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The world is full of data. The evolution of information science and technology has so explosively increased the amount of data that there is too much data for humans to analyze themselves. Therefore, humans have invented *machine learning*, which enables computers to learn, and machine learning now plays an important role in the advanced information society. Machine learning is expected to continue to develop as computers develop.

Machine learning is roughly classified into two types of learning depending on how training examples are presented: batch learning and online learning. A batch learning system is first given a large number of examples and then learns them all at once. In contrast, an online learning system is given examples sequentially and learns them one by one. In this dissertation, we address online classification problems. An online learning system is sequentially presented with training examples. The system outputs a class prediction, $H(\boldsymbol{x}_t)$, for the given input, $\boldsymbol{x}_t \in X$, at each time t and then updates its hypothesis, $H : X \to Y$, based on the true class label, $y_t \in Y$. Each example is independently drawn from the distribution of the target concept, $\Pr(\boldsymbol{x}, y)$. The task of the online learning system is to minimize cumulative prediction errors during online learning.

Many excellent systems have been proposed for batch learning, such as decision

trees [11, 82], support vector machines [13, 95], and bagging and boosting [9, 28, 65, 86]. However, there are serious problems with online learning, especially in environments where the statistical properties of the target variable may change over time, namely $\Pr_{t+1}(\boldsymbol{x}, y)$ may differ from $\Pr_t(\boldsymbol{x}, y)$. This change is known as *concept drift* [93, 99] and includes both gradual changes and sudden and significant changes. One example of concept drift is the spam filtering problem. Spam is characterized by skewed and changing class distributions, changing target concepts, and intelligent and adaptive adversaries [26, 66]. An effective spam filter must be able to adapt to the resulting changes.

There are many fields in which an effective system for handling concept drift is needed, and the number of studies that assume the presence of concept drift is growing. Representative early learning systems capable of handling concept drift are STAGGER [87], IB3 [2], and FLORA [98, 99]. More recently, many systems have been proposed to handle a wide variety of data types (e.g., vowel recognition [97], flight simulation [35], web page access [41], credit card transactions [96], electricity market data [31, 34], spam filtering [22], and text classification [47, 89, 100]). Concept drift has also been theoretically analyzed [4, 37, 52, 62].

An online learning system should be able to respond to both sudden and gradual changes. The effectiveness of strategies for building such systems depends on the types of changes, so creating an ideal system for various concept drift environments is difficult [53, 93]. For example, systems that have a sliding window and only adapt to training examples in the window are able to respond to gradual changes. However, such systems often fail to respond to sudden changes because the learning should be restarted from scratch if sudden and significant changes occur.

The online learning system should also be able to recognize and handle recurring concepts. An example of a recurring concept is Christmas-related spam, which is sent every December [26]. If a concept has appeared before, previous successful classifiers should be used. Using many classifiers built from old concepts is one way to handle

recurring concepts.

Moreover, the online learning system should be able to distinguish between true concept drift and noise. Misdetection of concept drift and overreaction to noise cause online learning systems to perform poorly, so the online learning system should combine sensitivity to concept drift with robustness to noise.

The online learning system should also be able to detect the occurrence of concept drift. The detection of concept drift helps the system respond quickly to sudden changes. There are also real problems to which change detection is relevant (e.g., user modeling, monitoring in biomedicine and industrial processes, fault detection and diagnosis, and safety of complex systems) [5, 31].

Therefore, an online learning system that can handle concept drift should be able to: (1) respond to both sudden and gradual changes; (2) recognize and treat recurring concepts effectively; (3) have both robustness to noise and sensitivity to true change; and (4) detect the occurrence of change. No such system has yet been developed. In this study, we conducted interdisciplinary studies as stepping-stones to develop the ideal system.

The rest of this dissertation is organized as follows. We first describe conventional online and batch learning systems, which are devoted to static environments, in Chapter 2. In Chapter 3, we present the *a*daptive *c*lassifiers-*e*nsemble (ACE) system, which focuses especially on handling recurring concepts. In Chapter 4, we describe the STEPD method, which uses a *s*tatistical *t*est of *e*qual *p*roportions to *d*etect concept drift, and then the *t*wo *o*nline classifiers for learning and *d*etecting concept dr*i*ft (Todi) system, which learns and detects concept drift quickly and accurately. In Chapter 5, we present the *l*eaky *i*ntegrate-and-*d*etect (LID) model, which is inspired by human behavior. Finally, we summarize the dissertation in Chapter 6.

Chapter 2

Fundamentals

In this chapter, we briefly explain several excellent machine learning systems that address classification problems. As we mentioned before, machine learning is roughly classified into two types of learning based on how training examples are presented: batch learning and online learning.

We first review some online learning systems, which are given examples sequentially and learn them one by one, in Section 2.1. We then explain some batch learning systems, which are first given a large number of examples and then learn them all at once, in Section 2.2. All of these excellent systems presented in this chapter are devoted to static environments and thus do not take concept drift into consideration. Our proposed systems and many other systems for learning concept drift use these excellent systems as their base classifiers.

2.1 Online Learning Systems

We first explain Naive Bayes and k-Nearest Neighbors. They are able to perform the same way in online and batch learning manners and thus achieve high predictive accuracy. We then describe the HMRAN ($h$yper $m$inimal $r$esource-$a$llocating $n$etwork) algorithm that we have previously proposed. This algorithm achieves fast

Algorithm 2.1 Naive Bayes classifier for discrete attributes.

1: **Initializes** N, N_y, $\forall N_{y,x_i} \leftarrow 0$.
2: **for** each training example $(\boldsymbol{x} = \{x_1, x_2, \ldots, x_M\}, y \in Y)$ **do**
3: **Output** $\arg\max_{y \in Y} \frac{N_y}{N} \prod_{i=1}^{M} \frac{N_{y,x_i}}{N_y}$.
4: increment N.
5: increment N_y.
6: **for** $i \in \{1, 2, \ldots, M\}$ **do**
7: increment N_{y,x_i}.
8: **end for**
9: **end for**

online learning by constructing itself autonomously.

2.1.1 Naive Bayes

The Naive Bayes classifier provides a simple approach, with clear semantics, to representing, using, and learning probabilistic knowledge [23, 44, 68]. The Naive Bayes classifier generally applies to learning tasks where each input $\boldsymbol{x}$ is described by a conjunction of discrete attribute values, $\{x_1, x_2, \ldots, x_M\}$.

From Bayes' theorem, the probability that an input $\boldsymbol{x} \in X$ belongs in class $y \in Y$ is

$$\Pr(Y = y | X = \boldsymbol{x}) = \frac{\Pr(Y = y) \Pr(X = \boldsymbol{x} | Y = y)}{\Pr(X = \boldsymbol{x})}. \tag{2.1}$$

We can estimate $\Pr(Y = y)$ easily by counting the number of training examples in class y, N_y, and the number of training examples seen so far, N. However, estimating $\Pr(X = \boldsymbol{x} | Y = y)$ in this fashion is not feasible unless we have quite a large number of training examples, because we need to see every possible example many times in order to obtain reliable estimates. Then, estimating $\Pr(X = \boldsymbol{x})$ is unnecessary because it is the same for all classes.

The Naive Bayes classifier assumes that all input attributes are independently given

Algorithm 2.2 Naive Bayes classifier for numeric attributes.

1: **Initializes** N, N_y, $\forall r_{y,i}$, $\forall s_{y,i} \leftarrow 0$.
2: **for** each training example $(\boldsymbol{x} = \{x_1, x_2, \ldots, x_M\}, y \in Y)$ **do**
3: $\mu_{y,i} \leftarrow r_{y,i}/N_y$.
4: $\sigma_{y,i} \leftarrow s_{y,i}/N_y - \mu_{y,i}^2$.
5: **Output** $\arg\max_{y \in Y} \frac{N_y}{N} \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi}\sigma_{y,i}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma_{y,i}^2}\right)$.
6: increment N.
7: increment N_y.
8: **for** $i \in \{1, 2, \ldots, M\}$ **do**
9: $r_{y,i} \leftarrow r_{y,i} + x_i$.
10: $s_{y,i} \leftarrow s_{y,i} + x_i^2$.
11: **end for**
12: **end for**

a class. We can therefore rewrite $\Pr(X = \boldsymbol{x}|Y = y)$ as

$$\Pr(X = \boldsymbol{x}|Y = y) = \prod_{i=1}^{M} \Pr(X_i = x_i|Y = y), \tag{2.2}$$

where we can estimate $\Pr(X_i = x_i|Y = y)$ easily by counting the number of training examples in class y having x_i, N_{y,x_i}.

To summarize, the Naive Bayes classifier outputs the class that maximizes

$$\Pr(Y = y) \prod_{i=1}^{M} \Pr(X_i = x_i|Y = y) = \frac{N_y}{N} \prod_{i=1}^{M} \frac{N_{y,x_i}}{N_y}. \tag{2.3}$$

Moreover, John and Langley presented a method that enables the Naive Bayes classifier to treat numeric attributes [44]. In the method, it is assumed that the value of each numeric attribute is normally distributed within each class. We can therefore write $\Pr(X_i = x_i|Y = y)$ as

$$\Pr(X_i = x_i|Y = y) = \frac{1}{\sqrt{2\pi}\sigma_{y,i}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma_{y,i}^2}\right), \tag{2.4}$$

where we can estimate $\mu_{y,i}$ and $\sigma_{y,i}$ by storing the summation of the values of ith

attribute x_i within class y and the squared summation of the values.

To summarize, the Naive Bayes classifier for numeric attributes outputs the class that maximizes

$$\frac{N_y}{N} \prod_{i=1}^{M} \frac{1}{\sqrt{2\pi}\sigma_{y,i}} \exp\left(-\frac{(x_i - \mu_{y,i})^2}{2\sigma_{y,i}^2}\right). \tag{2.5}$$

John and Langley also proposed a method, Flexible Bayes [44], which uses a set of kernels by estimating $\Pr(X_i = x_i | Y = y)$. However, it must store all of the examples seen so far.

2.1.2 k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm is the most basic instance-based method [19, 20]. k-NN is also a lazy learning method where it does not decide how to generalize beyond the training examples until each new input is encountered.

The training phase of k-NN consists of storing training examples only. In the classification phase, k-NN computes distances from the new input to all stored examples and chooses k closest examples. For numeric attributes, the distance is usually defined in terms of the standard Euclidean distance. For Boolean and discrete attributes, the distance is usually defined in terms of the number of attributes that two instances do not have in common [2]. k-NN then outputs the most common class among the k nearest neighbors. This k is a positive integer, typically small. If we choose $k = 1$, then the 1-NN algorithm outputs the class y' of the example $(\boldsymbol{x}', y')$, where $\boldsymbol{x}'$ is the nearest neighbor to the new input attributes $\boldsymbol{x}$.

k-NN is robust to noisy training examples and quite effective when it is provided a sufficiently large set of training examples, but storing all of the training examples significantly increases the computational cost to find k nearest neighbors. To solve this problem, memory indexing [6, 30], which enables k-NN to search nearest neighbors

Algorithm 2.3 k-nearest neighbors algorithm.

1: **Initializes** buffer $\boldsymbol{B} \leftarrow \Phi$.
2: **for** each training example $(\boldsymbol{x}, y \in Y)$ **do**
3: find k examples $\{\boldsymbol{x}_i, y_i\}_{i=1}^{k}$ that are nearest to $\boldsymbol{x}$ in $\boldsymbol{B}$.
4: **Output** $\arg\max_{y' \in Y} \sum_{i=1}^{k} [\![y' = y_i]\!]$.
5: add ($\boldsymbol{x}$, y) to $\boldsymbol{B}$.
6: **end for**

quickly, and editing and condensing [21], which reduce the number of stored examples, are used.

2.1.3 HMRAN

A resource-allocating network (RAN) that was proposed by Platt [78] is a neural network that allocates a new hidden unit whenever an unusual pattern is presented to the network. RAN achieves fast online learning, and there are several extensions of RAN [39, 40, 45, 60], including a minimal resource-allocating network (MRAN) [103]. MRAN is a Gaussian radial basis function (GRBF) network with a growing strategy and a pruning strategy for hidden units.

We proposed an online learning algorithm, the HMRAN (Hyper MRAN) algorithm [74, 77], by extending MRAN. The first extension is that HMRAN uses hyper basis functions (HyperBFs) instead of isotropic Gaussian basis functions to ignore useless input dimensions. Moreover, HMRAN uses a localized extended Kalman filter (LEKF) [7] instead of the global EKF that MRAN uses in order to reduce computational complexity. HMRAN also introduces a new merging strategy in addition to the pruning strategy of MRAN to avoid the generation of redundant hidden units. HMRAN achieves fast online learning and dimension selection with fewer resources. In this section, we describe the components of HMRAN briefly. Note that the main purpose of HMRAN is to solve regression problems, but HMRAN can also be used for classification problems. Algorithms 2.4 through 2.7 show the pseudo-code of HMRAN.

Algorithm 2.4 Hyper minimal resource-allocating network (HMRAN).

Parameters: $e_{\min}$, $\epsilon_{\max}$, $\epsilon_{\min}$, γ, κ, $e'_{\min}$, M, p_0. ($+$ q, λ, M_p, δ_p, f_p, M_m, δ_m)
for each example ($\boldsymbol{x}_t \in X, y_t \in Y$) **do**
  Output $\boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{c}_0 + \sum_{k=1}^{K} \boldsymbol{c}_k \phi_k(\boldsymbol{x}_t)$ with error $\boldsymbol{e}_t = \boldsymbol{y}_t - \boldsymbol{f}(\boldsymbol{x}_t)$.
  $\epsilon_t \leftarrow \max\left[\epsilon_{\max}\gamma^t, \epsilon_{\min}\right]$,
  $d_t \leftarrow \begin{cases} \min_k ||\boldsymbol{x}_t - \boldsymbol{t}_k||_{\boldsymbol{V}_k} & (K \neq 0) \\ \epsilon_{\max} & (K = 0) \end{cases}$,
  $e_{\mathrm{rms}t} \leftarrow \sqrt{(\sum_{i=t-(M-1)}^{t} ||\boldsymbol{e}_i||^2)/M}$.
  if $||\boldsymbol{e}_t|| > e_{\min}$ and $d_t > \epsilon_t$ and $e_{\mathrm{rms}t} > e'_{\min}$ **then** // allocate a new hidden unit
    $\boldsymbol{c}_{K+1} \leftarrow \boldsymbol{e}_t$, $\boldsymbol{t}_{K+1} \leftarrow \boldsymbol{x}_t$, $\boldsymbol{V}_{K+1} \leftarrow \boldsymbol{I}$, $\sigma_{K+1} \leftarrow \kappa d_t$.
    $\boldsymbol{P}_t = \begin{bmatrix} \boldsymbol{P}_{t-1} & \boldsymbol{0} \\ \boldsymbol{0} & p_0 \boldsymbol{I} \end{bmatrix}$. // adjust $\boldsymbol{P}_t$'s dimensionality to suit the increased network.
  else
    update network parameters using GEKF/LEKF.
  end if
  call pruning strategy.
  call merging strategy.
end for

A Hyper Basis Function Network

Let $\boldsymbol{x}_t$ be the input at time t, $\phi_k(\boldsymbol{x}_t)$ be the output of the kth hidden unit, $\boldsymbol{c}_k$ be the link weight vector for the output layer, and c_0 be the bias term. The output of the network is expressed as $\boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{c}_0 + \sum_{k=1}^{K} \boldsymbol{c}_k \phi_k(\boldsymbol{x}_t)$. MRAN uses a Gaussian as a basis function, $\phi_k(\boldsymbol{x}_t) = \exp\left(-||\boldsymbol{x}_t - \boldsymbol{t}_k||^2/\sigma_k^2\right)$, where $\boldsymbol{t}_k$ and σ_k are the center of and the width of the kth hidden unit; whereas HMRAN uses a hyper basis function [80] as the basis function of the kth hidden unit to ignore useless input dimensions,

$$\phi_k(\boldsymbol{x}_t) = \exp\left(-||\boldsymbol{x}_t - \boldsymbol{t}_k||^2_{\boldsymbol{V}_k}/\sigma_k^2\right), \tag{2.6}$$

where $||\cdot||_{\boldsymbol{V}_k}$, which is called the weighted norm, is expressed as

$$||\boldsymbol{x}_t - \boldsymbol{t}_k||^2_{\boldsymbol{V}_k} = (\boldsymbol{x}_t - \boldsymbol{t}_k)^T \boldsymbol{V}_k^T \boldsymbol{V}_k (\boldsymbol{x}_t - \boldsymbol{t}_k). \tag{2.7}$$

To reduce computational complexity, HMRAN uses a diagonal matrix, $\boldsymbol{V}_k = \text{diag}\{\boldsymbol{v}_k\}$, where $\boldsymbol{v}_k = [v_{k_1}, v_{k_2}, ..., v_{k_J}]^T$. This diag$\{\boldsymbol{v}_k\}$ denotes the matrix that contains $\boldsymbol{v}_k$ in diagonal elements and 0 in all other elements. The value of v_{k_j} indicates the importance of the jth dimension in the kth hidden unit.

B Growing Strategy

HMRAN begins the learning of the target function without hidden units. A new hidden unit is allocated to the network if all of the following three conditions are satisfied:

$$||\boldsymbol{e}_t|| = ||\boldsymbol{y}_t - \boldsymbol{f}(\boldsymbol{x}_t)|| > e_{\min}, \tag{2.8}$$

$$d_t = \begin{cases} \min_k ||\boldsymbol{x}_t - \boldsymbol{t}_k|| & (K \neq 0), \\ \epsilon_{\max} & (K = 0) \end{cases} > \epsilon_t, \text{ and} \tag{2.9}$$

$$e_{\text{rms}t} = \sqrt{\sum_{i=t-(M-1)}^{t} ||\boldsymbol{e}_i||^2 / M} > e'_{\min}, \tag{2.10}$$

where ϵ_t is given at time t by $\epsilon_t = \max[\epsilon_{\max}\gamma^t, \epsilon_{\min}]$ $(0 < \gamma < 1)$. The parameters for the new hidden unit are set to $\boldsymbol{c}_{K+1} = \boldsymbol{e}_t$, $\boldsymbol{t}_{K+1} = \boldsymbol{x}_t$, $\boldsymbol{V}_{K+1} = \boldsymbol{I}$, and $\sigma_{K+1} = \kappa d_t$, where κ is called the overlap factor and determines the overlap of the hidden units' responses in the input space. If one of these conditions is not satisfied, HMRAN updates its network parameters.

C Updating Network Parameters

The global EKF that MRAN (GEKF) uses would need a high computational cost for updating network parameters. Birgmeier described an algorithm for updating GRBF parameters that applies an extended Kalman filter (EKF) method [36] independently to each hidden unit [7]. HMRAN offers not only the GEKF but this localized EKF (LEKF) to users by means of reducing the computational cost.

Algorithm 2.5 Localized extended Kalman filter (LEKF) for HMRAN.

Parameters: q, λ.
// updating network parameters with the current example $(\boldsymbol{x}_t, y_t)$. //////////
// $\boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{c}_0 + \sum_{k=1}^{K} \boldsymbol{c}_k \phi_k(\boldsymbol{x}_t)$; $\boldsymbol{e}_t \leftarrow \boldsymbol{y}_t - \boldsymbol{f}(\boldsymbol{x}_t)$.
for each kth hidden unit **do**
  $\boldsymbol{w}_{t-1}^k \leftarrow [\boldsymbol{t}_k^T, \boldsymbol{v}_k^T, \sigma_k]^T$, $\boldsymbol{\mu}_t^k \leftarrow (2\phi_k/{\sigma_k}^2)\boldsymbol{c}_k$,
  $\boldsymbol{\nu}_t^k \leftarrow [(\boldsymbol{x}_t - \boldsymbol{t}_k)^T \boldsymbol{V}_k^T \boldsymbol{V}_k,\ -(\boldsymbol{x}_t - \boldsymbol{t}_k)^T \mathrm{diag}\{\boldsymbol{x}_t - \boldsymbol{t}_k\}\boldsymbol{V}_k^T,\ ||\boldsymbol{x}_t - \boldsymbol{t}_k||^2_{\boldsymbol{V}_k}/\sigma_k]^T$,
  $\boldsymbol{\psi}_t^k \leftarrow \boldsymbol{P}_{t-1}^k \boldsymbol{\nu}_t^k$, $\alpha_t^k \leftarrow {\boldsymbol{\nu}_t^k}^T \boldsymbol{\psi}_t^k$, $\beta_t^k \leftarrow {\boldsymbol{\mu}_t^k}^T \boldsymbol{\mu}_t^k$.
  update $\boldsymbol{w}_t^k \leftarrow \boldsymbol{w}_{t-1}^k + \dfrac{{\boldsymbol{\mu}_t^k}^T \boldsymbol{e}_t}{\lambda + \alpha_t^k \beta_t^k} \boldsymbol{\psi}_t^k$, $\boldsymbol{P}_t^k \leftarrow \boldsymbol{P}_{t-1}^k - \dfrac{\beta_t^k}{\lambda + \alpha_t^k \beta_t^k} \boldsymbol{\psi}_n^k {\boldsymbol{\psi}_n^k}^T + q\boldsymbol{I}$.
end for
$\boldsymbol{s}_t \leftarrow [1, \phi_1, \phi_2, ..., \phi_K]^T$.
for each lth output unit **do**
  $\boldsymbol{c}_{t-1}^l \leftarrow [c_{0_l}, c_{1_l}, c_{2_l} ..., c_{K_l}]^T$, $\boldsymbol{\xi}_t^l \leftarrow \boldsymbol{P}_{t-1}^l \boldsymbol{s}_t$, $\zeta_t^l \leftarrow \boldsymbol{s}_t^T \boldsymbol{\xi}_t^l$.
  update $\boldsymbol{c}_t^l \leftarrow \boldsymbol{c}_{t-1}^l + \dfrac{e_t^l}{\lambda + \zeta_t^l} \boldsymbol{\xi}_t^l$, $\boldsymbol{P}_t^l \leftarrow \boldsymbol{P}_{t-1}^l - \dfrac{1}{\lambda + \zeta_t^l} \boldsymbol{\xi}_t^l {\boldsymbol{\xi}_t^l}^T + q\boldsymbol{I}$.
end for

Algorithm 2.5 shows the pseudo-code of the LEKF that HMRAN uses. Note that we extended Birgmeier's localized EKF method to handle multidimensional outputs. The LEKF method has the advantage of requiring no matrix inversions, and thus reduces the computational cost more significantly than the global EKF (GEKF) method that MRAN uses.

D Pruning Strategy

HMRAN first calculates $\boldsymbol{o}_k(\boldsymbol{x}_n) = \boldsymbol{c}_k \exp\left(-||\boldsymbol{x}_n - \boldsymbol{t}_k||^2/\sigma_k^2\right)$ for each hidden unit. It then calculates the contribution rates of hidden units for each output unit,

$$r_{k_l} = |o_{k_l}/o_{\max_l}| \qquad (l = 1, \ldots, L), \tag{2.11}$$

where $\boldsymbol{o}_{\max}(\boldsymbol{x}_n) = [o_{\max_1}, \ldots, o_{\max_L}] = [\max_k |o_{k_1}|, ..., \max_k |o_{k_L}|]^T$.

If the contribution rates of the kth hidden unit for all output units are less than a threshold value, namely $(\forall l)\, r_{k_l} < \delta_p$, for the M_p consecutive observations, the kth hidden unit is pruned. The dimensionality of all related matrices is then adjusted to

Algorithm 2.6 Pruning strategy of HMRAN.

Parameters: M_p, δ_p, f_p.
if $||\boldsymbol{f}(\boldsymbol{x}_t)|| > f_p$ **then** // remove redundant units
  $\boldsymbol{o}_k(\boldsymbol{x}_t) \leftarrow \boldsymbol{c}_k \exp\left(-||\boldsymbol{x}_t - \boldsymbol{t}_k||^2_{\boldsymbol{V}_k}/{\sigma_k}^2\right) \quad (k = 1 \dots K)$,
  $\boldsymbol{o}_{\max}(\boldsymbol{x}_t) \leftarrow [\max_k |o_{k_1}|, ..., \max_k |o_{k_L}|]^T$.
  for each kth hidden unit **do**
    $r_{k_l} \leftarrow |o_{k_l}/o_{\max_l}| \quad (l = 1 \dots L)$.
    if $(\forall l) r_{k_l} < \delta_p$ for M_p consecutive observations **then**
      prune the kth hidden unit.
      adjust the dimensionality of $\boldsymbol{P}_t$ to suit the reduced network.
    end if
  end for
end if

suit the reduced network. To calculate the contribution rate accurately, HMRAN executes this pruning strategy procedure only if $||f(\boldsymbol{x}_t)|| > f_p$ is satisfied. Algorithm 2.6 shows the pseudo-code of the pruning strategy of HMRAN.

E Merging Strategy

First, HMRAN randomly selects a candidate unit for merging among hidden units, and then selects another candidate unit of which the center is the nearest to the center of the candidate unit. Next, HMRAN generates pseudo-examples around the centers of these two candidate units, and then updates the parameters of the single merged unit so that the output of the merged unit is equal to the sum of the outputs of the two hidden units. And finally, the two candidate units are merged into the merged unit, if the error rate for the pseudo-examples is less than a threshold value.

It is difficult for common online learning algorithms to get a sufficient number of training examples for merging. However, generating pseudo-examples enables HMRAN to accurately merge two units into a single unit.

Algorithm 2.7 Merging strategy of HMRAN.

Parameters: M_m, δ_m.
select $m_1 \in \{1, ..., K\}$ randomly; $m_2 \leftarrow \arg\min_k ||\boldsymbol{t}_k - \boldsymbol{t}_{m_1}||_{\boldsymbol{V}_{m_1}}$. // find two hidden units.
if $||\boldsymbol{t}_{m_2} - \boldsymbol{t}_{m_1}||_{\boldsymbol{V}_{m_1}} > \epsilon_n$ **then**
  return. // do not merge.
end if
for $i = 1$ to M_m **do** // generate pseudo patterns.
  select $k \in \{m_1, m_2\}$, $\rho_i \in [0, \epsilon_n]$, $u_{ij} \in [-1, 1]$ $(j = 1, ..., J)$ randomly.
  $\boldsymbol{x}_i^* \leftarrow \boldsymbol{t}_k + \rho_i \boldsymbol{u}_i / ||\boldsymbol{u}_i||$, $\boldsymbol{y}_i^* \leftarrow \boldsymbol{o}_{m_1}(\boldsymbol{x}_i^*) + \boldsymbol{o}_{m_2}(\boldsymbol{x}_i^*)$.
end for
$\boldsymbol{c}_m \leftarrow \boldsymbol{c}_{m_1} + \boldsymbol{c}_{m_2}$, $\boldsymbol{t}_m \leftarrow (\boldsymbol{t}_{m_1} + \boldsymbol{t}_{m_2})/2$, $\boldsymbol{v}_m \leftarrow (\boldsymbol{v}_{m_1} + \boldsymbol{v}_{m_2})/2$, $\sigma_m \leftarrow (\sigma_{m_1} + \sigma_{m_2})/2$. // initialize the parameters of unit m.
$h \leftarrow 0$.
repeat
  update $h \leftarrow h + 1$.
  calculate $E_h = 0.5 \sum_{i=1}^{M_m} ||\boldsymbol{y}_i^* - \boldsymbol{o}_m(\boldsymbol{x}_i^*)||^2$.
  update $\boldsymbol{w}_m \leftarrow \boldsymbol{w}_m - \eta \nabla_{\boldsymbol{w}_m} E_h$. // learning of the unit m's parameters: $\boldsymbol{w}_m$.
  if $|E_h - E_{h-1}| < \delta_s$ **then** // stop criterion.
    calculate $e_{\text{rms}}^* \leftarrow \sqrt{(\sum_{i=1}^{M_m} ||\boldsymbol{y}_i^* - \boldsymbol{o}_m(\boldsymbol{x}_i^*)||^2)/M_m}$.
    if $e_{\text{rms}}^* < \delta_m$ **then**
      calculate $o_{\text{rms}}^k \leftarrow \sqrt{(\sum_{i=1}^{M_m} ||\boldsymbol{o}_k(\boldsymbol{x}_i^*)||^2)/M_m}$ $(k = m_1, m_2)$.
      if $o_{\text{rms}}^{m_1} > o_{\text{rms}}^{m_2}$ **then**
        $s \leftarrow m_1$, $d \leftarrow m_2$.
      else
        $s \leftarrow m_2$, $d \leftarrow m_1$.
      end if
      $\boldsymbol{w}_s \leftarrow \boldsymbol{w}_m$. // to inherit the error covariance matrix.
      prune the dth hidden unit; adjust the dimensionality of $\boldsymbol{P}_n$ to suit the reduced network.
    end if
  end if
until $|E_h - E_{h-1}| < \delta_s$

2.2 Batch Learning Systems

We first explain decision trees and support vector machines. These two batch learning systems are powerful methods and are thus widely used for learning from a large amount of data. We then describe bagging and boosting that are well-known classifier

ensemble systems. These classifier ensemble systems are general methods for improving the accuracy of any given batch learning algorithm, which also includes decision trees and support vector machines. The ideas of bagging and boosting are often used in multiple classifier systems for learning concept drift.

2.2.1 Decision Trees

Decision tree learning is one of the most widely and practically used learning methods. Its representative algorithms are ID3 [81], C4.5 [82], and CART [11].

A decision tree consists of leafs, decision nodes, and branches. A leaf is a terminal node and indicates a class. A decision node is a non-terminal node. It specifies a test of an attribute and has a branch and a subtree for each possible outcome of the test.

An input is classified by starting at the root decision node of a decision tree. At each decision node, the input's outcome for the test is determined, and the input goes down the branch corresponding to this outcome. When this process finally leads to a leaf, the decision tree predicts the class that is indicated by the leaf.

Decision trees are constructed recursively. Decision tree algorithms try to select the best attribute (test) that divides a set of training examples into subsets so that each subset belongs to a single class. The algorithms then remove the selected attribute from further consideration, and then try to select the best attribute for each subset. If all training examples in the given set belong to a single class, the algorithms just return a leaf node of that class. Most decision tree algorithms use the criterion for attribute selection based on information entropy (e.g. gain ratio [82]).

2.2.2 Support Vector Machines

Support vector machines (SVMs) are one of the most powerful methods for solving classification problems [13, 95]. Several effective algorithms and tools for fast training

linear SVMs have been proposed [42, 43, 79].

SVMs belong to a family of linear classifiers. SVMs learn linear decision rules in their basic form,

$$f(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w}^T\boldsymbol{x} + b) = \begin{cases} +1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}, \tag{2.12}$$

where $\boldsymbol{w}$ is a weight vector and b is a threshold. In the classification phase, SVMs predict the positive class if $\boldsymbol{w}^T\boldsymbol{x} + b > 0$ and the negative class otherwise.

SVMs try to select the maximum-margin hyperplane from training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. That is to say, SVMs try to find the hyperplane so that the distance from the hyperplane to the nearest training examples of the positive and negative classes is maximized. If the training data are linearly separable, SVMs can select the hyperplane. The optimization problem of SVMs is a quadratic programming optimization problem:

$$\text{minimize } \frac{1}{2}||\boldsymbol{w}||^2, \text{ subject to } y_i(\boldsymbol{w}_i^T\boldsymbol{x}_i + b) \geq 1 \quad (i = 1, \ldots, N). \tag{2.13}$$

Moreover, this algorithm can be extended to non-linearly separable examples by using the soft margin method. Furthermore, SVMs can map training examples into a high-dimensional space by using a kernel function so that the problem is linearly separable. This technique is called the kernel trick, and it enables SVM to achieve non-linear classification and quite low generalization errors.

2.2.3 Bagging

Bagging (bootstrap aggregating) generates multiple bootstrap training sets from the original training set and uses each set to generate a base classifier in the ensemble [9]. The pseudo-code is shown in Algorithm 2.8.

Algorithm 2.8 Bagging.

1: **Parameters:** J: number of classifiers, $\boldsymbol{B} = \{\boldsymbol{x}_n \in X, y_n \in Y\}_{n=1}^{N}$: training set, D: distribution of examples.
2: **for** $j = 1$ to J **do**
3: set $\boldsymbol{B}_j \leftarrow \Phi$.
4: **for** $n = 1$ to N **do**
5: resample an example $(\boldsymbol{x}, y)$ from $\boldsymbol{B}$ randomly.
6: add $(\boldsymbol{x}, y)$ to $\boldsymbol{B}_j$.
7: **end for**
8: build base classifier $H_j : X \rightarrow Y$ from $\boldsymbol{B}_j$.
9: **end for**
10: **Output** $H(\boldsymbol{x}_n) = \arg\max_{y \in Y} \sum_{j=1}^{J} [\![H_j(\boldsymbol{x}_n) = y]\!]$.

Bagging creates J bootstrap training sets from original training set $\boldsymbol{B}$ and generates J classifiers using each set. These J training sets have some differences, and these differences cause variations among the J classifiers. If the classifiers are almost the same, the high predictive accuracy of the ensemble is not realized. Bagging uses a simple majority vote for determining the output class of the whole system.

2.2.4 Boosting

Boosting is based on the question of whether a weak learning algorithm, which performs just slightly better than random guessing, can be boosted into an arbitrarily accurate strong learning algorithm [29, 86]. AdaBoost was proposed by Freund and Schapire [28] and is the most popular boosting algorithm. The pseudo-code of Adaboost is shown in Algorithm 2.9. This is a version of AdaBoost for multi-classes, Adaboost.M1.

AdaBoost builds a given base classifier repeatedly in a series of rounds. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the classifier is forced to focus on the hard examples in the training set. The system output is a weighted majority

Algorithm 2.9 Adaboost.M1.

Parameters: J : number of iterations, $\boldsymbol{B} = \{\boldsymbol{x}_n \in X, y_n \in Y\}_{n=1}^{N}$: training set.
Initialize $\forall D_1(i) \leftarrow 1/N$.
for $j = 1$ to J **do**
  set $\boldsymbol{B}_j \leftarrow \Phi$.
  for $n = 1$ to N **do**
    resample an example $(\boldsymbol{x}, y)$ from $\boldsymbol{B}$ according to D_j.
    add $(\boldsymbol{x}, y)$ to $\boldsymbol{B}_j$.
  end for
  build base classifier $H_j : X \rightarrow Y$ from $\boldsymbol{B}_j$ with error $\epsilon_j = \Pr_{i \sim D_j}\left[H_j(\boldsymbol{x}_i) \neq y_i\right] = \sum_i D_j(i) \left[\!\left[H_j(\boldsymbol{x}_i) \neq y_i\right]\!\right]$.
  if $\epsilon_j > 1/2$ **then**
    abort loop.
  end if
  choose $\alpha_j = \frac{1}{2} \ln\left(\frac{1-\epsilon_j}{\epsilon_j}\right)$.
  update $\forall D_{j+1}(i) \leftarrow D_j(i) \exp\left(-\alpha_j \left(1 - 2\left[\!\left[H_j(\boldsymbol{x}_i) \neq y_i\right]\!\right]\right)\right)$.
  normalize $\forall D_{j+1}(i) \leftarrow \frac{D_{j+1}(i)}{\sum_k D_{j+1}(k)}$.
end for
Output $H(\boldsymbol{x}) = \arg\max_{y \in Y} \sum_{j=1}^{J} \alpha_j \left[\!\left[H_j(\boldsymbol{x}) = y\right]\!\right]$.

vote from the outputs of the base classifiers. The weight of the jth classifier, which measures the importance that is assigned to the classifier, is given by

$$\alpha_j = \frac{1}{2} \ln\left(\frac{1-\epsilon_j}{\epsilon_j}\right). \tag{2.14}$$

Also, ϵ_j is the error that is measured with respect to the distribution D_j on which the jth base classifier was trained.

Chapter 3

Learning Concept Drift with Multiple Classifiers

Using multiple classifiers would be a natural solution to learning concept drift. Several systems have been proposed that use a majority weighted vote from the outputs of base classifiers for learning concept drift as shown in Section 3.1. To overcome drawbacks that conventional systems have, we proposed the *a*daptive *c*lassifiers-*e*nsemble (ACE) system that we describe in Section 3.2. The ACE system consists of a single online classifier, many batch classifiers, and a drift detection mechanism. ACE especially focuses on handling recurring concepts, and experimental results show that ACE was able to handle recurring concepts better than conventional systems. However, the original ACE system does not have any classifier pruning method, and its original weighting method, which combines the outputs of all base classifiers, does not reduce interference from old classifiers sufficiently. We have therefore added a classifier pruning method and we have improved the original weighting method. We present the enhanced ACE system in Section 3.3. Experimental results show that the enhanced ACE system was better able to learn concept drift than the original ACE system. Finally, we summarize this study in Section 3.4.

3.1 Background and Related Work

Most of the current research in multiple classifier systems is devoted to static environments [9, 10, 28, 65, 86]. However, several systems have recently been proposed that use ensembles of classifiers for learning concept drift [24, 25, 49, 50, 89–92, 96]. Responding to various types of concept drift is difficult for a single classifier, so the number of studies using multiple classifier systems has been growing. Most of the systems build an ensemble of classifiers from sequential data streams and use a majority vote from the outputs of up-to-date base classifiers.

In this section, we describe some conventional classifier ensemble systems for learning concept drift. We first present two batch classifier ensemble systems that use ensembles of batch classifiers built from sequential chunks of training examples, the streaming ensemble algorithm (SEA) [92] and Wang et al.'s classifier ensemble approach (WCEA) [96]. We then present two online classifier ensemble systems that use ensembles of online classifiers trained with new training example at each time, the dynamic weighted majority (DWM) algorithm [49] and the AddExp algorithm [50].

3.1.1 Streaming Ensemble Algorithm

The streaming ensemble algorithm (SEA) was proposed by Street and Kim [92]. SEA uses a simple majority vote from the outputs of batch classifiers built from sequential chunks of training examples. It responds to concept drift by replacing an unnecessary classifier with a new one. The pseudo-code of SEA is shown in Algorithm 3.1. They thought that simple accuracy is not the best criterion and proposed a quality criterion for replacing classifiers in the ensemble. This criterion is based on the outputs of base classifiers, as shown in Algorithm 3.2 .

SEA performs well in gradually changing environments but has trouble responding

Algorithm 3.1 Streaming ensemble algorithm (SEA).

Parameters: C: chunk size, M: capacity of classifiers
Initialize buffer $\boldsymbol{B} \leftarrow \Phi$, number of classifiers $J \leftarrow 0$.
while more data point $(\boldsymbol{x}_t, y_t)$ is available **do** // $\boldsymbol{x}_t \in X$, $y_t \in Y$
  get classifiers output $\{H_j(\boldsymbol{x}_t)\}_{j=1}^{J-1} \in Y$.
  Output $H(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=1}^{J-1} [\![H_j(\boldsymbol{x}_t) = y]\!]$.
  add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$.
  if $|\boldsymbol{B}| \geq C$ **then**
    build batch classifier $H_{J+1} : X \rightarrow Y$ from $\boldsymbol{B}$.
    if $J < M$ **then**
      $J \leftarrow J + 1$. // insert H_J into ensemble.
    else if Quality$(H_J, \boldsymbol{B}) >$ min$_j$ Quality$(H_j, \boldsymbol{B})$ **then**
      $j' \leftarrow \arg\min_j$ Quality(H_j).
      replace $H_{j'}$ with H_J.
    end if
    initialize $\boldsymbol{B}$.
  end if
end while

Algorithm 3.2 Quality(H_k) function of SEA.

Parameters: H_k: kth hypothesis, $\boldsymbol{B}$: examples.
Initialize weights $w = 0$.
for each $(\boldsymbol{x}_i, y_i) \in \boldsymbol{B}$ **do**
  get classifiers output $\{H_j(\boldsymbol{x_i})\}_{j=1}^{J}$ and the system output $H(\boldsymbol{x}_i)$.
  $P_1 \leftarrow$ the proportion of votes $\{H_j(\boldsymbol{x_i})\}_{j=1}^{J}$ for the majority class.
  $P_2 \leftarrow$ the proportion of votes for the second-most voted for class.
  $P_c \leftarrow$ the proportion for the correct class y_i.
  $P_i \leftarrow$ the proportion for the class $H_i(\boldsymbol{x}_i)$.
  if $H(\boldsymbol{x}_i) = y_i$ and $H_k(\boldsymbol{x}_i) = y_i$ **then**
    $w \leftarrow w + 1 - |P_1 - P_2|$.
  else if $H(\boldsymbol{x}_i) \neq y_i$ and $H_k(\boldsymbol{x}_i) = y_i$ **then**
    $w \leftarrow w + 1 - |P_1 - P_c|$.
  else if $H_k(\boldsymbol{x}) \neq y_i$ **then**
    $w \leftarrow w + 1 - |P_i - P_c|$.
  end if
end for
return w.

to sudden changes. If a sudden change occurs, incorrect outputs of classifiers from old concepts interfere with the outputs of classifiers for new ones. SEA therefore performs poorly for sudden changes.

Algorithm 3.3 Wang et al.'s classifier ensemble approach (WCEA).

Parameters: C: chunk size, M: capacity of classifiers
Initialize buffer $\boldsymbol{B} \leftarrow \Phi$, number of classifiers $J \leftarrow 0$.
while more data point $(\boldsymbol{x}_t, y_t)$ is available **do** // $\boldsymbol{x}_t \in X$, $y_t \in Y$
  get classifiers output $\forall_y \{H_j^y(\boldsymbol{x}_t)\}_{j=1}^J \in [0,1]$.
  Output $H(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=1}^J w_j H_j^y(\boldsymbol{x}_t) [\![w_j > 0]\!]$.
  add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$.
  if $|\boldsymbol{B}| \geq C$ **then**
    build batch classifier $H_{J+1} : X \rightarrow Y$ from $\boldsymbol{B}$.
    $\mathrm{MSE}_{J+1} \leftarrow$ error rate of H_{J+1} via the cross validation method on $\boldsymbol{B}$.
    $\mathrm{MSE}_r \leftarrow$ error rate of the random classifiers on $\boldsymbol{B}$.
    $w_{J+1} \leftarrow \mathrm{MSE}_r - \mathrm{MSE}_{J+1}$.
    for $j = 1$ to J **do**
      $\mathrm{MSE}_j \leftarrow \dfrac{1}{|\boldsymbol{B}|} \displaystyle\sum_{(\boldsymbol{x},y) \in \boldsymbol{B}} (1 - H_j^y(\boldsymbol{x}))^2$
      $w_j \leftarrow \mathrm{MSE}_r - \mathrm{MSE}_j$.
    end for
    if $J < M$ **then**
      $J \leftarrow J + 1$.
    else if $w_{J+1} > \min_{j=1}^J w_j$ **then**
      $j' \leftarrow \arg\min_{j=1}^J w_j$.
      replace H_j' with H_{J+1}.
    end if
    initialize $\boldsymbol{B}$.
  end if
end while

3.1.2 Wang et al.'s Classifier Ensemble Approach

Wang et al. proposed a classifier ensemble approach similar to SEA for mining concept drift data streams [96]. In this approach, which we refer to as WCEA, a weighted average is used to combine the classifier outputs. The weight of each classifier is the error rate of a random classifier minus the mean square error of the classifier for the current chunk. The mean square errors of old classifiers are high, and thus the weights of old classifiers are small. While WCEA also performs well in gradually changing environments, it cannot reduce the interference that arises from sudden changes unless small chunks are used. This is because the weights of all classifiers do not change until

Algorithm 3.4 Dynamic weighted majority (DWM) algorithm.

Parameters: $\beta \in [0, 1)$: factor for decreasing weights, p: period between classifier removal, creation, and weight update
Initialize number of experts $J = 1$, expert weight $w_1 = 1$.
while more data point $(\boldsymbol{x}_t, y_t)$ is available **do** // $\boldsymbol{x}_t \in X$, $y_t \in Y$
  get classifiers output $\{H_j(\boldsymbol{x}_t)\}_{j=1}^{J} \in Y$.
  Output $H_f(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=1}^{J} w_j \, [\![H_j(\boldsymbol{x}_t) = y]\!]$.
  if $n \mod p = 0$ **then**
    update $\forall w_j \leftarrow w_j \beta^{[\![H_j(\boldsymbol{x}_t) \neq y_t)]\!]}$.
    normalize $\forall w_j = w_j / \max_i[w_i]$.
    if $H_f(\boldsymbol{x}_t) \neq y_t$ **then**
      add a new classifier: $w_{J+1} = 1$, $J \leftarrow J + 1$.
    end if
  end if
  train all classifiers $\forall H_j : X \rightarrow Y$ with $(\boldsymbol{x}_t, y_t)$.
end while

the next chunk is given. Unfortunately, smaller chunks worsen the predictive accuracy of each classifier and thus of the overall system. Therefore, small chunks cannot be used for solving quite difficult problems. Furthermore, SEA and WCEA remove the most unnecessary classifier at the time of pruning, so they often cannot respond to a recurring concept quickly even if they had previously learned the concept. Although many batch classifier ensemble systems have been proposed [24, 25, 47, 89], these problems have not been solved.

3.1.3 Dynamic Weighted Majority

Kolter and Maloof proposed a system that uses an ensemble of online classifiers for learning concept drift. The dynamic weighted majority (DWM) system is based on the weighted majority algorithm [61], but they added mechanisms to add and remove classifiers dynamically in response to changes in performance. DWM maintains an ensemble of base online classifiers and predicts a class for the current input by using a weighted majority vote from the outputs of the base classifiers. The DWM algorithm is shown in Algorithm 3.4.

Algorithm 3.5 AddExp for discrete class prediction.

Parameters: $\beta \in [0, 1]$: factor for decreasing weights, $\gamma \in [0, 1]$: factor for new expert weight
Initialize number of experts $J = 1$, expert weight $w_1 = 1$.
while more data point $(\boldsymbol{x}_t, y_t)$ is available **do** // $\boldsymbol{x}_t \in X$, $y_t \in Y$
  get classifiers output $\{H_j(\boldsymbol{x}_t)\}_{j=1}^{J} \in Y$.
  Output $H(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=1}^{J} w_j \, [\![H_j(\boldsymbol{x}_t) = y]\!]$.
  update $\forall w_j \leftarrow w_j \beta^{[\![H_j(\boldsymbol{x}_t) \neq y_t]\!]}$.
  if $H(\boldsymbol{x}_t) \neq y_t$ **then**
    add a new classifier: $w_{J+1} \leftarrow \gamma \sum_{j=1}^{J} w_j$, $J \leftarrow J + 1$.
  end if
  if $J > K$ **then**
    prune the oldest or weakest classifier.
  end if
  train all classifiers $\forall H_j : X \rightarrow Y$ with $(\boldsymbol{x}_t, y_t)$.
end while

If a base classifier predicts incorrectly, then its weight is reduced by the multiplicative constant β. If the system output is not correct, then DWM adds a new online classifier with a weight of one. DWM normalizes classifier weights by uniformly scaling them so that the highest weight will be equal to one. This prevents any newly added classifiers from dominating the decision-making of existing ones. DWM removes classifiers with weights that are less than a threshold θ. Large and noisy problems require the parameter p, which governs the frequency by which DWM adds classifiers, removes classifiers, and updates weights.

3.1.4 AddExp

Kolter and Maloof then proposed the additive expert ensemble algorithm, AddExp [50]. The AddExp system is simpler than DWM. AddExp adds a new classifier whenever the system output is not correct. Also, AddExp sets the weight of a new classifier to the total weight of the ensemble times a constant γ. Moreover, they proposed two pruning methods: oldest first and weakest first. In the oldest first method, the oldest classifier is removed if the number of classifiers is greater than some constant. In

contrast, the classifier with the lowest weight is removed in the weakest first method. They mentioned that the weakest first strategy performs well in practice because it removes the worst performing classifiers.

After sudden and significant changes, the weights of classifiers corresponding to old concepts are reduced quickly because the outputs of such classifiers are often not correct for the current concept. AddExp can therefore discard old classifiers quickly and respond to sudden changes quickly. However, base classifiers are apt to forget the content of learning from old concepts because the base classifiers are updated at all times. Moreover, DWM and AddExp also do not consider the occurrence of recurring concepts.

3.2 The ACE system

To overcome the drawbacks that the conventional systems have, we have proposed an online learning system that uses an ensemble of classifiers for learning concept drift. We had originally reported the *a*daptive *c*lassifiers-*e*nsemble (ACE) system in [75] and then revised it slightly [70, 76]. In this section, we present the original ACE system. We first outline ACE and describe the system's components: a single online classifier, many batch classifiers, a drift detection mechanism, a sliding window, and a long-term buffer. We then describe a weighting method, which combines the outputs of base classifiers. ACE focuses especially on handling recurring concepts, and we demonstrate that ACE was better able to handle recurring concepts than the conventional systems. Finally, we briefly summarize the study of the original ACE system. The pseudo-code of the original ACE system is shown in Algorithm 3.6.

3.2.1 Outline

The basic concept of ACE is illustrated in Figure 3.1. ACE behaves as follows:

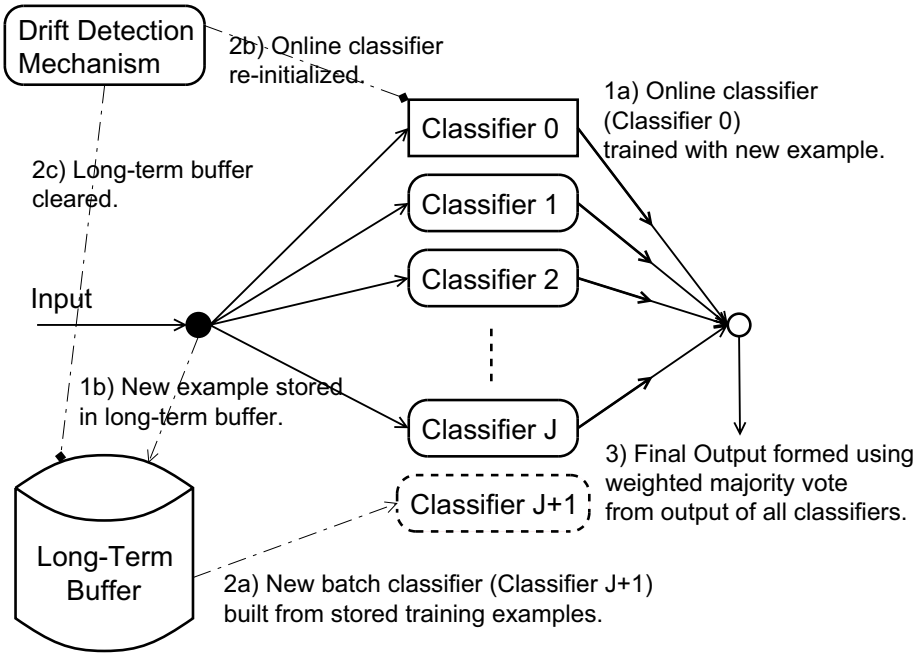

Fig. 3.1 Basic concept of ACE.

1. Let $(\boldsymbol{x}_t \in X, y_t \in Y)$ be a new training example at time t. ACE trains the online classifier, $H_0 : X \to Y$, with the new example. The new example is stored in a long-term buffer, $\boldsymbol{B}$.
2. A new, J+1th, batch classifier, $H_{J+1} : X \to Y$, is built from stored examples, when concept drift is detected or the number of stored examples exceeds the capacity of the buffer. The J denotes the current number of batch classifiers. And then, the online classifier is reinitialized and the long-term buffer is cleared. These enable the online classifier to learn the next concept from scratch. Note that the batch classifiers are not updated.
3. The system output is given by a weighted majority vote from the outputs of all base classifiers. The predictive accuracy of each classifier at time t for the most recent W training examples, $\{A_{j,t}\}_{j=1}^{J}$, is used for deciding the weights.

Algorithm 3.6 Original adaptive classifiers-ensemble (ACE) system.

Parameters: W: window size, L: capacity of long-term buffer, $1-\alpha$: confidence level, M: capacity of classifiers.
Initialize number of batch classifiers $J \leftarrow 0$, long-term buffer $\boldsymbol{B} \leftarrow \Phi$.
while more data point ($\boldsymbol{x}_t \in X$, $y_t \in Y$) is available **do**
  get classifiers outputs $\{H_j(\boldsymbol{x}_t)\}_{j=0}^{J} \in Y$.
  Output $H(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=0}^{J} (1 - A_{j,t-1})^{-\mu} [\![H_j(\boldsymbol{x}_t) = y]\!]$.
  for $j = 0$ to J **do** // for each classifier
    $R_{j,t} \leftarrow [\![H_j(\boldsymbol{x}_t) = y_t]\!]$. // result.
    $A_{j,t} \leftarrow \begin{cases} \sum_{s=t-|B|+1}^{t} R_{j,s}/(|\boldsymbol{B}|+1) & \text{if } j=0 \ \&\ |\boldsymbol{B}| < W \\ \sum_{s=t-W+1}^{t} R_{j,s}/(W+1) & \text{otherwise} \end{cases}$. // accuracy.
    $A_{j,t}^{l,\alpha} \leftarrow \frac{W}{W+z_{\alpha/2}^2} \left(A_{j,t} + \frac{z_{\alpha/2}^2}{2W} - z_{\alpha/2} \sqrt{\frac{A_{j,t}(1-A_{j,t})}{W} + \frac{z_{\alpha/2}^2}{4W^2}} \right)$. // confidence interval.
    $A_{j,t}^{u,\alpha} \leftarrow \frac{W}{W+z_{\alpha/2}^2} \left(A_{j,t} + \frac{z_{\alpha/2}^2}{2W} + z_{\alpha/2} \sqrt{\frac{A_{j,t}(1-A_{j,t})}{W} + \frac{z_{\alpha/2}^2}{4W^2}} \right)$.
  end for
  train online classifier $H_0 : X \to Y$ with ($\boldsymbol{x}_t$,y_t).
  add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$.
  // Adding the new J+1th classifier. ////////////////////////////
  $j^* \leftarrow \arg\max_{j=1 \ldots J} \sum_{s=t-W+1}^{t} A_{j,s}$.
  Addition $\leftarrow$ **false**.
  if $J = 0$ and $|\boldsymbol{B}| \geq 2W$ **then**
    Addition $\leftarrow$ **true**. // to build first batch classifier.
  else if $|\boldsymbol{B}| \geq L$ **then**
    Addition $\leftarrow$ **true**. // to periodically build new batch classifier.
  else if $|\boldsymbol{B}| \geq 2W$ and $\{A_{j^*,t} < A_{j^*,t-W}^{l,\alpha}$ or $A_{j^*,t} > A_{j^*,t-W}^{u,\alpha}\}$ **then**
    Addition $\leftarrow$ **true**. // concept drift occurred.
  end if
  if *Addition* = **true then**
    build batch classifier from $\boldsymbol{B}$, with $H_{J+1} : X \to Y$.
    // copy records from online classifier to new classifier.
    $\{R_{J+1}\} \leftarrow \{R_0\}$, $\{A_{J+1}\} \leftarrow \{A_0\}$, $\{A_{J+1}^{l,\alpha}\} \leftarrow \{A_0^{l,\alpha}\}$, and $\{A_{J+1}^{u,\alpha}\} \leftarrow \{A_0^{u,\alpha}\}$.
    $J \leftarrow J + 1$.
    reinitialize H_0, $\{R_0\}$, $\{A_0\}$, $\{A_0^{l,\alpha}\}$, $\{A_0^{u,\alpha}\}$.
    $\boldsymbol{B} \leftarrow \Phi$.
  end if
end while

3.2.2 Online Classifier

The response of batch classifier ensemble systems to sudden changes is delayed until a new chunk is given. We therefore added an online classifier that is trained with one

example at each time. We can use any online learning algorithm for the online classifier.

3.2.3 Batch Classifiers

When concept drift is detected or the number of stored examples exceeds the capacity of the long-term buffer, ACE builds a new batch classifier from the training examples stored in the buffer. We can use any batch learning algorithm for each batch classifier. Note that ACE does not update the previously built batch classifiers.

3.2.4 Drift Detection Mechanism

There has been much work on detecting changes [5, 8, 12, 14, 63, 64], but most of it is based on estimating the underlying distribution of training examples. Although these detection methods depend on the type of input attributes and require a large number of examples, our proposed method does not depend on the type of input attributes and is able to detect concept drift in a small number of examples. Concept drift is detected by observing the predictive accuracy of each classifier for recent training examples. The predictive accuracy of the jth classifier at time t for the most recent W training examples is defined as

$$A_{j,t} = \frac{\left(\sum_{s=t-W+1}^{t} R_{j,s}\right) + 1}{W+2} = \frac{S_{j,t}+1}{W+2}, \tag{3.1}$$

where $R_{j,t} = [\![H_j(\boldsymbol{x}_t) = y_t]\!]$ is the classification result of the jth classifier at time t, and $[\![H_j(\boldsymbol{x}_t) = y_t]\!]$ is 1 if $H_j(\boldsymbol{x}_t) = y_t$ is true and 0 otherwise. Note that the classification result of the online classifier, $R_{0,t}$, is the result before the classifier is trained with the newest example, $(\boldsymbol{x}_t, y_t)$. Equation (3.1) is Laplace's rule of succession [57] and achieves a better estimation of the successful classification rate than simply dividing $S_{j,t}$ by W (maximum likelihood estimate) and by $W+1$ (the original definition [75]),

especially when sample sizes (W) are small [59].

Confidence intervals of the predictive accuracies of batch classifiers are used to detect concept drift. The upper endpoint, $A^{u,\alpha}_{j,t}$, and the lower endpoint, $A^{l,\alpha}_{j,t}$, of the $100(1-\alpha)\%$ confidence interval for the predictive accuracy of the jth classifier are given by

$$A^{l,\alpha}_{j,t} = \frac{W}{W+z^2_{\alpha/2}}\left(A_{j,t}+\frac{z^2_{\alpha/2}}{2W}-z_{\alpha/2}\sqrt{\frac{A_{j,t}(1-A_{j,t})}{W}+\frac{z^2_{\alpha/2}}{4W^2}}\right) \text{ and} \tag{3.2}$$

$$A^{u,\alpha}_{j,t} = \frac{W}{W+z^2_{\alpha/2}}\left(A_{j,t}+\frac{z^2_{\alpha/2}}{2W}+z_{\alpha/2}\sqrt{\frac{A_{j,t}(1-A_{j,t})}{W}+\frac{z^2_{\alpha/2}}{4W^2}}\right), \tag{3.3}$$

where $z_{\alpha/2}$ is the upper $(100\alpha/2)$th percentile of the standard normal distribution [38].

If $A_{j,t}$ is greater than $A^{u,\alpha}_{j,t-W}$ or less than $A^{l,\alpha}_{j,t-W}$, concept drift is suspected because ACE does not update the batch classifiers.

We defined the index of the best classifier at time t as

$$j^* = \arg\max_{j=1\ldots J}\sum_{s=n-W+1}^{n} A_{j,s}, \tag{3.4}$$

and if

$$|\boldsymbol{B}| \geq 2W \text{ and } \{A_{j^*,t} < A^{l,\alpha}_{j^*,t-W} \text{ or } A_{j^*,t} > A^{u,\alpha}_{j^*,t-W}\}, \tag{3.5}$$

ACE assumes that concept drift occurred (see Figure 3.2) and builds the new J+1th classifier. The $|\boldsymbol{B}|$ denotes the number of training examples stored in the long-term buffer. To detect subsequent concept drift properly, this method needs $2W$ examples: W examples to calculate $A_{j^*,t-W}$ and $[A^{l,\alpha}_{j^*,t-W}, A^{u,\alpha}_{j^*,t-W}]$, and the other W examples to calculate $A_{j^*,t}$.

Checking only the predictive accuracy of the best classifier is more effective for avoiding false alarms than checking the predictive accuracies of all batch classifiers.

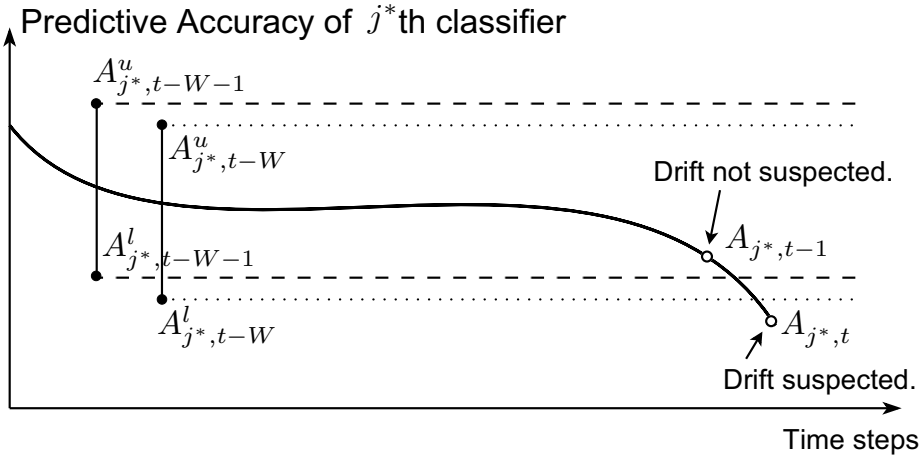

Fig. 3.2 Concept drift detection by ACE.

Generally, the predictive accuracies of classifiers corresponding to a concept are often not stable for other concepts.

Moreover, ACE builds a new batch classifier when the number of stored training examples exceeds the capacity of the long-term buffer, even if concept drift is not suspected, as in the WCEA system. In addition, to build the first batch classifier rapidly, ACE builds it when the number of batch classifiers is zero and the number of stored training examples exceeds twice the sliding window size, $2W$. Furthermore, to avoid fluctuations in the predictive accuracy of ACE immediately after the addition of a new classifier, ACE copies the records of the online classifier stored in the sliding window, $\{R_{0,m}, A_{0,m}, A^{l,\alpha}_{0,m}, A^{u,\alpha}_{0,m}\}^{n}_{m=n-W-1}$, to those of the new classifier, $\{R_{J+1,m}, A_{J+1,m}, A^{l,\alpha}_{J+1,m}, A^{u,\alpha}_{J+1,m}\}^{n}_{m=n-W-1}$, when ACE adds the new classifier.

3.2.5 Sliding Window

The sliding window stores the classification results of, the predictive accuracy of, and the confidence intervals of each classifier for the most recent W training examples. Each classifier therefore does not have to reclassify past training examples when its predictive accuracy is updated. Moreover, ACE does not have to recalculate the past confidence intervals of each classifier when ACE is checking for concept drift.

3.2.6 Long-Term Buffer

The long-term buffer is used to store recent training examples and to build batch classifiers. To be able to respond to sudden changes quickly, batch classifiers ensemble systems have to set a small chunk size because its weights do not change until the next chunk is given. However, small chunks worsen the predictive accuracy of each classifier and thus of the overall system. ACE can respond to sudden changes even if we set the buffer large because ACE updates its weights at each time and has a drift detection mechanism.

3.2.7 Weighting Method

ACE forms the system output by using a weighted majority vote from the outputs of all base classifiers. The system output of ACE is:

$$H(\boldsymbol{x}_t) = \arg\max_{y \in Y} \sum_{j=0}^{J} (1 - A_{j,t})^{-\mu} [\![H_j(\boldsymbol{x}_t) = y]\!], \tag{3.6}$$

where μ is an adjustment factor of the ensemble ($\mu \geq 0$).

Using $(1 - A_{j,t})^{-\mu}$ for the weighted majority vote avoids interference from old classifiers and leverages prior knowledge of recurring concepts. Here, a larger μ approximates the system output by using the selection of the best classifier for recent training examples, whereas a smaller μ approximates the system output by using the simple majority vote.

3.2.8 Experimental Results

We created a concept drift situation by using the STAGGER concepts, a standard benchmark for concept drift [35, 49, 50, 87, 99], to provide empirical support for the

performance of ACE for recurring concepts.

The problem consists of three concepts for a domain of figures described by *color* ∈ {red, green, blue}, *shape* ∈ {circle, square, triangle}, and *size* ∈ {small, medium, large}. The three concepts are (1) [*size* = small] and [*color* = red], (2) [*color* = green] or [*shape* = circle], and (3) [*size* = medium or large]. The presentation of training examples lasts for 900 time steps with the target concept changing every 100 examples, in the cyclical order (1)-(2)-(3)-(1)-(2)-(3)-(1)-(2)-(3). At each time, an online learning system was trained with one example and then was tested with 100 examples generated randomly according to the current concept.

We compared ACE with Naive Bayes, Naive Bayes trained only on each concept (Naive Bayes reinitialized every 100 examples), SEA, WCEA, DWM, and AddExp. The parameter settings for ACE were $W = 17$, $L = 109$, $\alpha = 0.2$, and $\mu = 3.0$. The chunk size and the capacity of base classifiers for SEA were 19 and 3, and those for WCEA were 29 and 20[†3.1]. WCEA used a 5-fold cross validation method. The factor for decreasing weights and the threshold for deleting classifiers for DWM and AddExp were 0.5 and 0.01. DWM also adds a new classifier whenever the system output is not correct. The capacity of base classifiers for AddExp was 10. AddExp used the weakest first pruning method. The base online classifier of ACE, DWM, and AddExp was Naive Bayes and the base batch classifier of ACE, SEA, and WCEA was C4.5 [82]. Note that SEA and WCEA classify input data randomly when the number of their base classifiers is zero. The predictive accuracy of and the number of classifiers for each system are shown in Figures 3.3 and 3.4. All results were averaged over 100 trials.

To respond to sudden changes quickly, batch classifier ensemble systems (SEA and WCEA) need small chunks. Also, SEA needs a small capacity of base classifiers because SEA uses a simple majority vote from the outputs of all base classifiers. We made the chunk size and the capacity of the classifiers quite small; however, the responses of

[†3.1] The chunk sizes of ACE, SEA, and WCEA were determined so that the time at which they add a new classifier from the chunk of data does not exactly coincide with the time of concept drift.

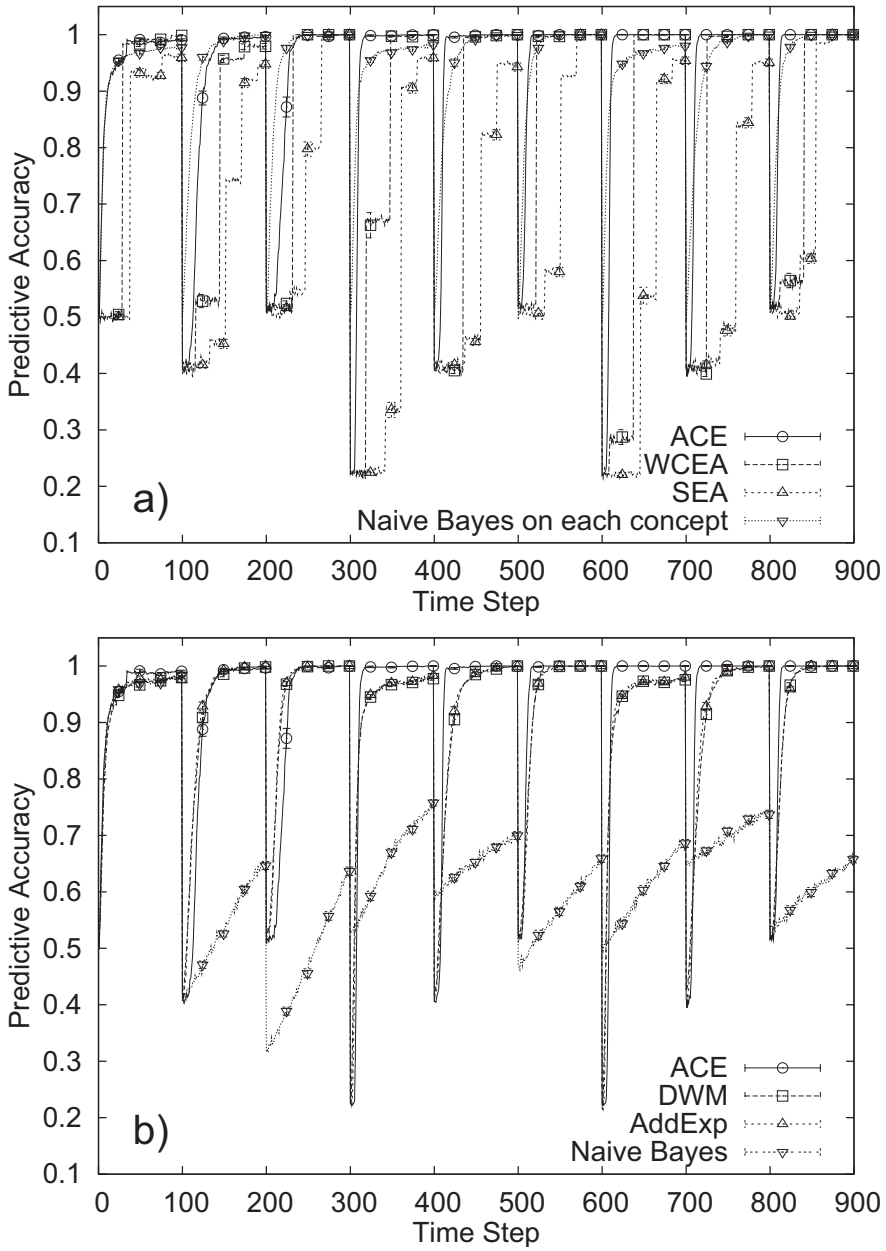

Fig. 3.3 Predictive accuracies on the STAGGER concepts. (a) ACE, SEA, WCEA, and Naive Bayes trained only on each concept. (b) ACE, DWM, AddExp, and Naive Bayes. The error bars indicate standard errors from 100 trials.

SEA to changes were delayed, as shown in Figure 3.3. On the other hand, WCEA can store more classifiers without worsening the predictive accuracy because it uses an accuracy-based weighted majority vote. Thus, WCEA can use many classifiers for handling recurring concepts; however, WCEA was not able to respond to sudden

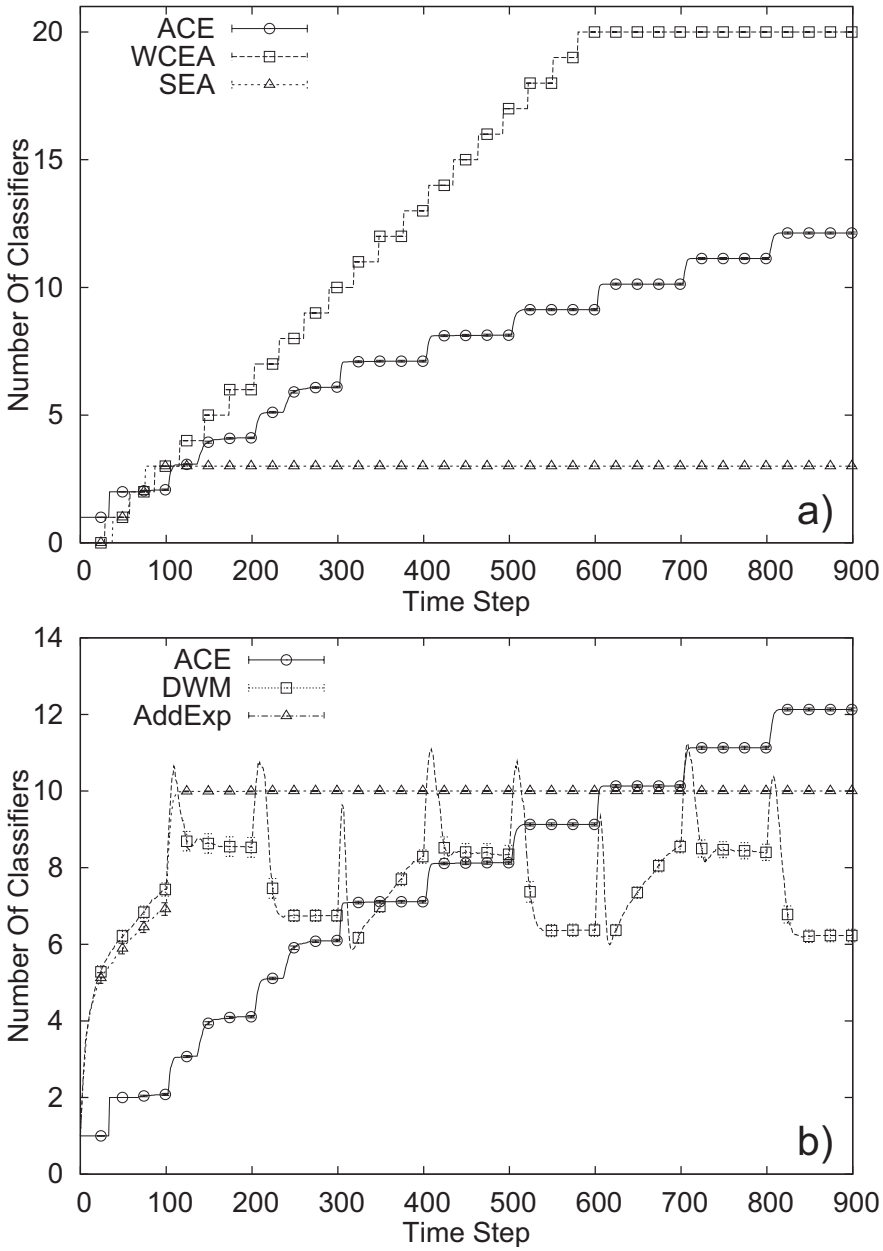

Fig. 3.4 Numbers of classifiers on the STAGGER concepts. (a) ACE, SEA, and WCEA. (b) ACE, DWM, and AddExp. The error bars indicate standard errors from 100 trials.

changes quickly because its weights do not change until the next chunk is given. Using a drift detection mechanism and an online classifier enabled ACE to respond to sudden changes quickly. Then, we can see that the online classifier ensemble systems (DWM and AddExp) were able to respond to sudden changes quickly. However, the systems

do not have classifiers that are not updated, so the systems have problems handling recurring concepts. Using both online and batch classifiers enabled ACE to handle recurring concepts well in the second and third presentation of each concept (from time 301 to 900). Moreover, from Figure 3.4, we can see that ACE gave false alarms during the first presentation of each concept. ACE is apt to give false alarms if it does not have any batch classifiers for the current concept. However, in this experiment, the false alarm caused ACE to build the new batch classifier for the current concept quickly and thus to achieve a high predictive accuracy.

3.2.9 Short Summary

We proposed an online learning system that uses an ensemble of classifiers for tracking concept drift. This adaptive classifiers-ensemble system, ACE, consists of a single online classifier, many batch classifiers, and a drift detection mechanism. Its weighting method, which combines the outputs of base classifiers, uses the predictive accuracy of each classifier for recent training examples in order to avoid interference between the outputs of classifiers from old concepts and classifiers for new ones. The drift detection mechanism uses confidence intervals for the predictive accuracies of batch classifiers for recent training examples. ACE focuses especially on handling recurring concepts, and experimental results showed that ACE was better able to handle recurring concepts than conventional systems.

The original ACE system that we have presented in this Section 3.2 does not have any classifier pruning method. Storing many classifiers is essential for handling recurring concepts, but it may cause interference with learning new concepts. Moreover, the original weighting method does not reduce the interference from old classifiers sufficiently. We therefore proposed the enhanced ACE system described in the next Session 3.3.

3.3 The Enhanced ACE system

We have added a new classifier pruning method and have improved the original weighting method [71, 72] to overcome the drawbacks that the original ACE system has. In this section, we first present the new features of the enhanced ACE system: the new classifier pruning method, which enables ACE to remove redundant classifiers, and the improved weighting method, which enables ACE to sufficiently reduce interference from old classifiers. We then demonstrate that the enhanced ACE system performed well in learning concept drift for both synthetic and actual datasets. Finally, we briefly summarize the study of the enhanced ACE system. The pseudo-code of the enhanced ACE system is shown in Algorithm 3.7.

3.3.1 Classifier Pruning Method

Storing many classifiers is essential for handling recurring concepts. However, it may cause interference with learning new concepts, even with the improved weighting method presented in the next Session 3.3.2. Moreover, if redundant classifiers are not removed, memory resources become overtaxed and computational complexity grows significantly. We thus need an effective classifier pruning method.

The weakest first method of AddExp and the pruning method of WCEA use a single set of weights both to combine the outputs of base classifiers and to remove the classifier with the lowest weight. These methods do not consider the occurrence of recurring concepts, so they often remove classifiers that should be kept for later reuse. To handle recurring concepts, ACE uses two sets of weights: one to decide the system output of ACE, $\{v_j\}$, and the other one to determine the order of pruning, $\{w_j\}$.

ACE sets the initial weight of each new classifier, w_{J+1}, to zero. At each time, the

Algorithm 3.7 Enhanced adaptive classifiers-ensemble (ACE) system.

Parameters: W: window size, L: capacity of long-term buffer, $1-\alpha$, $1-\beta$: confidence levels, M: capacity of classifiers.
Initialize no. of batch classifiers $J=0$, long-term buffer $\boldsymbol{B}=\Phi$.
while more data point ($\boldsymbol{x}_n \in X$, $y_n \in Y$) is available **do**
  get classifiers outputs $\{H_j(\boldsymbol{x}_n)\}_{j=0}^{J} \in Y$.
  Output $H(\boldsymbol{x}_n) = \arg\max_{y\in Y} \sum_{j=0}^{J} v_j \, [\![H_j(\boldsymbol{x}_n) = y]\!]$
  where $v_j = \begin{cases} \log\left(\frac{A_{j,t-1}}{1-A_{j,t-1}}\right) & \text{if } A_{j,t-1} > \max_j A_{j,t-1}^{l,\beta} \\ 0, & \text{otherwise.} \end{cases}$
  for $j=0$ to J **do** // for each classifier
    $R_{j,t} \leftarrow [\![H_j(\boldsymbol{x}_t) = y_t]\!]$. // result.
    $A_{j,t} \leftarrow \begin{cases} (\sum_{s=n-|B|+1}^{n}(R_{j,s}+1)/(|\boldsymbol{B}|+2) & \text{if } j=0 \;\&\; |\boldsymbol{B}|<W \\ (\sum_{s=n-W+1}^{n}(R_{j,s}+1)/(W+2) & \text{otherwise} \end{cases}$. // accuracy.
    $A_{j,t}^{l,\alpha} \leftarrow \frac{W}{W+z_{\alpha/2}^2}\left(A_{j,t} + \frac{z_{\alpha/2}^2}{2W} - z_{\alpha/2}\sqrt{\frac{A_{j,t}(1-A_{j,t})}{W} + \frac{z_{\alpha/2}^2}{4W^2}}\right)$. // confidence interval.
    $A_{j,t}^{u,\alpha} \leftarrow \frac{W}{W+z_{\alpha/2}^2}\left(A_{j,t} + \frac{z_{\alpha/2}^2}{2W} + z_{\alpha/2}\sqrt{\frac{A_{j,t}(1-A_{j,t})}{W} + \frac{z_{\alpha/2}^2}{4W^2}}\right)$.
  end for
  train online classifier $H_0 : X \to Y$ with $(\boldsymbol{x}_n, y_n)$.
  add $(\boldsymbol{x}_n, y_n)$ to $\boldsymbol{B}$.
  // update weights for pruning. ////////////////////////////
  $j_{\max} \leftarrow \arg\max_{j=0\ldots J} A_{j,t}$.
  $w_j \leftarrow \begin{cases} w_j+1 & \text{if } j=j_{\max} \text{ and } w_j \geq 0 \\ 0 & \text{if } j=j_{\max} \text{ and } w_j<0 \\ w_j-1 & \text{otherwise} \end{cases} \quad (j=1,\ldots,J).$
  // Adding the new J+1th classifier. ////////////////////////////
  $j^* \leftarrow \arg\max_{j=1\ldots J} \sum_{s=n-W+1}^{n} A_{j,s}$.
  Addition $\leftarrow$ **false**.
  if $J=0$ and $|\boldsymbol{B}| \geq 2W$ **then**
    Addition $\leftarrow$ **true**. // to build first batch classifier.
  else if $|\boldsymbol{B}| \geq L$ **then**
    Addition $\leftarrow$ **true**. // to periodically build new batch classifier.
  else if $|\boldsymbol{B}| \geq 2W$ and $\{A_{j^*,t} < A_{j^*,t-W}^{l,\alpha}$ or $A_{j^*,t} > A_{j^*,t-W}^{u,\alpha}\}$ **then**
    Addition $\leftarrow$ **true**. // concept drift occurred.
  end if
  if *Addition* = **true then**
    build batch classifier from $\boldsymbol{B}$, with $H_{J+1} : X \to Y$.
    // copy records from online classifier to new classifier.
    $\{R_{J+1}\} \leftarrow \{R_0\}$, $\{A_{J+1}\} \leftarrow \{A_0\}$, $\{A_{J+1}^{l,\alpha}\} \leftarrow \{A_0^{l,\alpha}\}$, and $\{A_{J+1}^{u,\alpha}\} \leftarrow \{A_0^{u,\alpha}\}$.
    if $J+1 > M-1$ **then**
      remove H_k where $k = \arg\min_{j=1}^{J} w_j$.
    else
      $J \leftarrow J+1$.
    end if
    reinitialize H_0, $\{R_0\}$, $\{A_0\}$, $\{A_0^{l,\alpha}\}$, $\{A_0^{u,\alpha}\}$.
    $\boldsymbol{B} \leftarrow \Phi$.
  end if
end while

weight of each batch classifier is updated:

$$w_j \leftarrow \begin{cases} w_j+1 & \text{if } j = j_{\max} \;\&\; w_j \geq 0 \\ 0 & \text{if } j = j_{\max} \;\&\; w_j < 0 \\ w_j-1 & \text{otherwise} \end{cases} \quad (j=1,\ldots,J), \tag{3.7}$$

where the index of the batch classifier with the highest predictive accuracy for the most recent W training examples is

$$j_{\max} = \arg \max_{j=0\ldots J} A_{j,t}. \tag{3.8}$$

Consequently, the weight of the $j_{\max}$th batch classifier is equal to or greater than zero. To prevent the storing of old classifiers only, the weights of classifiers are decreased by one at each time, except for the $j_{\max}$th classifier. Note that if $j_{\max} = 0$ then the weights of all of the batch classifiers are decreased by one.

If the number of classifiers exceeds the capacity of base classifiers, the kth batch classifier with the lowest weight is removed, where

$$k = \arg \min_{j=1\ldots J} w_j. \tag{3.9}$$

Older classifiers are thus removed first, unless they have the highest predictive accuracy over a long period of time. Note that the online classifier is never removed.

3.3.2 Improved Weighting Method

The enhanced ACE system forms its output by using this improved weighted majority vote from the outputs of all classifiers:

$$H(\boldsymbol{x}_t) = \arg \max_{y \in Y} \textstyle\sum_{j=0}^{J} v_j \left[\!\left[H_j(\boldsymbol{x}_t) = y \right]\!\right], \tag{3.10}$$

where v_j is

$$v_j = \begin{cases} \log\left(\dfrac{A_{j,t-1}}{1-A_{j,t-1}}\right) & \text{if } A_{j,t-1} > \max_j A^{l,\beta}_{j,t-1} \\ 0, & \text{otherwise} \end{cases}. \tag{3.11}$$

The $A^{l,\beta}_{j,t-1}$ is the lower endpoint of the $100(1-\beta)\%$ confidence interval of the predictive accuracy of the jth classifier for the most W training examples.

With the improved weighting method, Equation (3.11), ACE is able to reduce interference from old classifiers sufficiently and to only use adaptive classifiers in the ensemble for deciding the system output. ACE uses an AdaBoost-like function that is often used for a classifier ensemble as the weighting function, v_j [28].

3.3.3 Experiments and Results

We used two datasets in the experiments. The first one is a synthetic dataset, which is a modified SEA concepts [92], and the second one is an actual spam filtering dataset, which is the TREC 2005 Spam Track Public Corpus [18]. We demonstrate that the enhanced ACE system performed better than the original ACE system in learning concept drift.

A Modified SEA concepts

We created a concept drift situation by using SEA concepts, which is a benchmark for concept drift [49, 92]. To evaluate the performance in more difficult situations, we modified the original problem so that it has 30 attributes, $\boldsymbol{x}_t \in [0, 10]^{30}$. The target concept was $y = \left[\!\left[\sum_{i=1}^{6} x_{t,i} \leq \theta\right]\!\right]$ where b is a threshold value. Thus, only the first six attributes are relevant. Furthermore, noise was introduced by switching the labels of 10% of the training examples. We used threshold values of $b = 30 - (10/9200)t$, 27, 38, 21, and 27 for eight data blocks. Each block lasted 2300 time steps. The threshold

changed gradually from 30 to 20 during the first four blocks ($b = 30 - (10/9200)t$; $0 \leq t < 9200$) and then changed suddenly during the last four blocks ($b = 27$, 38, 21, and 27). Consequently, recurring concepts appeared in the fifth block ($b = 27$), the seventh block ($b = 21$), and the eighth block ($b = 27$). The presentation of training examples lasted for 18400 (2300×8) time steps. At each time, an online learning system was trained with one example and then was tested with 200 examples generated randomly according to the current concept. We compared the enhanced ACE with the original ACE, AddExp, and WCEA to evaluate the overall effectiveness of the enhanced ACE. The parameter settings for the ACEs were $W = 200$, $L = 1200$, $\alpha = 0.999$, $\beta = 0.80$, and $\mu = 2.0$. The factor for decreasing weights and the factor for a new classifier weight for AddExp were 0.5 and 0.01. AddExp used the weakest first method. The chunk size for WCEA was 100. The base classifiers for the ACEs, AddExp, and WCEA were Naive Bayes that can handle numeric attributes [44, Equations (1) – (3)]. Each system, except the original ACE, had a capacity of 12 classifiers. Note that the original ACE does not remove any classifiers. All results were averaged over 500 trials.

As shown in Figure 3.5, both ACEs were able to respond to both sudden and gradual changes and recurring concepts. The new pruning method enabled the enhanced ACE to remove redundant classifiers without significant degradation in the predictive accuracy compared to the original ACE that does not remove any classifiers. The improved weighting method enabled the enhanced ACE to quickly respond to the sudden change at the beginning of the sixth block where the threshold value was novel ($b = 38$). This was because the improved weighting method is able to effectively reduce interference from old classifiers that have low predictive accuracy. AddExp was able to respond to sudden changes quickly and accurately. However, in the third and fourth blocks, it worked poorly because old classifiers were not replaced with new classifiers quickly enough for the gradual changes. Moreover, AddExp had problems handling recurring concepts because it does not have any base classifiers that are not updated. WCEA performed better than AddExp for gradual changes but performed poorly for sud-

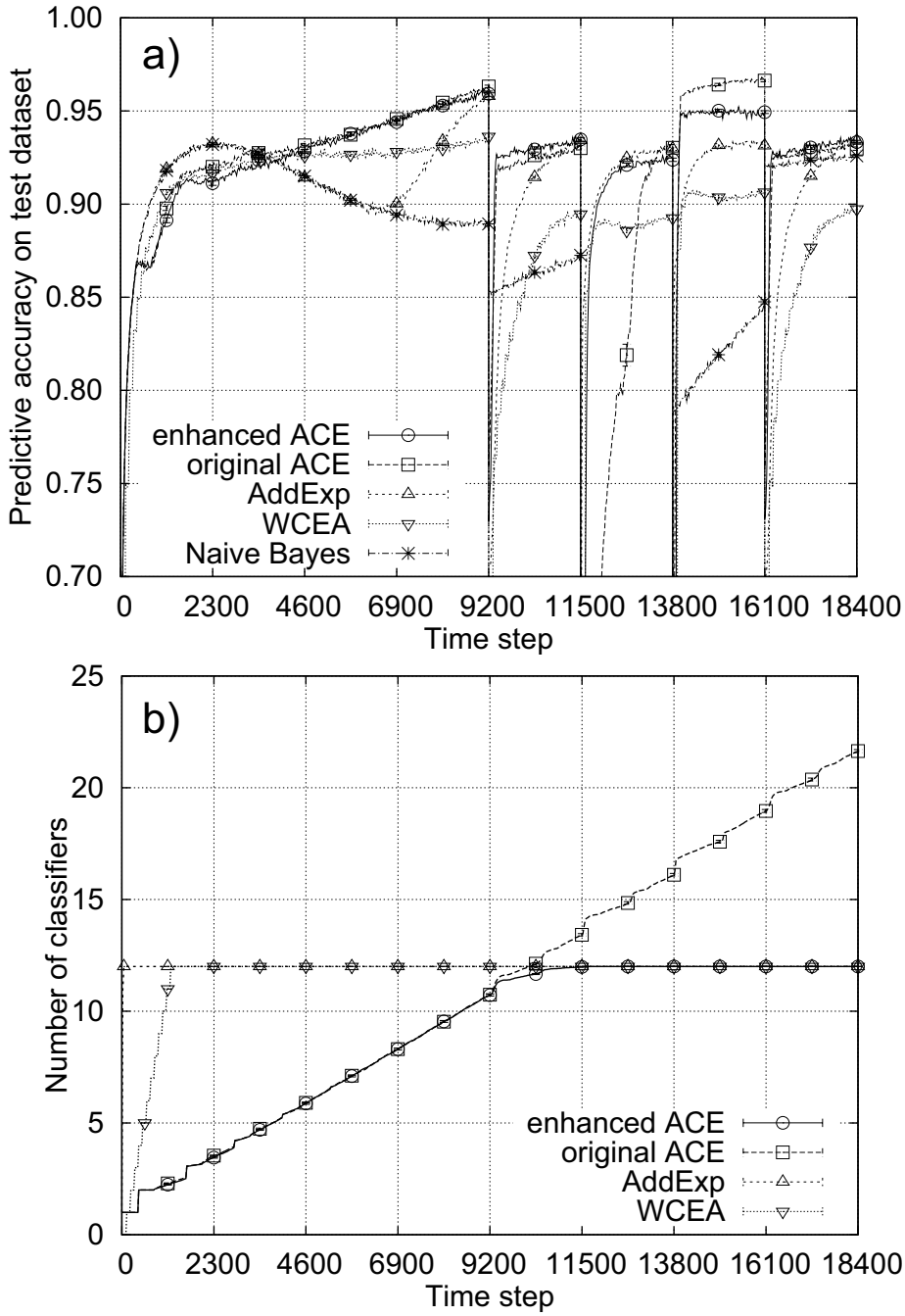

Fig. 3.5 (a) Predictive accuracies and (b) numbers of classifiers of the enhanced ACE, the original ACE, AddExp, and WCEA on the modified SEA concepts. The error bars indicate standard errors of mean from 500 trials.

den changes. WCEA was not able to reduce the interference that arises from sudden changes unless small chunks were used. However, smaller chunks worsened the predictive accuracy of each classifier and thus of the overall system. Furthermore, WCEA

had problems handling recurring concepts because it removed the most unnecessary classifier at the time of pruning.

B TREC 2005 Spam Track Public Corpus

In the second experiment, we evaluated the effectiveness of the new features in the enhanced ACE system and its overall performance using an actual dataset, the TREC 2005 Spam Track Public Corpus [18], for situations where there is concept drift. The dataset that we used, trec05p-1/full, has 92,189 raw messages in chronological order for the period June 29, 2001 to June 29, 2002, of which about 57% are spam [17]. Spam filtering is problematic because the topics of and the characteristics of messages shift over time, and the spam ratio fluctuates wildly (see Figure 3.6).

We treated numbers as letters and treated punctuation marks and other non-alphabetic characters as separate tokens. For example, `ab123cd@example.com` was tokenized as `<ab123cd> <@> <example> <.> <com>`. We did not remove any headers, HTML tags, and stop words[†3.2]. Moreover, we did not perform any stemming[†3.3]. The purpose of this experiment was to evaluate the effectiveness of classifier ensemble systems for tracking concept drift in actual data, not to determine whether ACE is an effective spam filter. We selected 100 attributes (tokens) from the first 1000 messages (from June 29, 2001 to July 10, 2001), which had a spam ratio of about 81%, by using information gain [82]. The other 91189 messages were presented to online learning systems, and the learning systems were trained with one message converted to the 100 Boolean attributes at each time. To confirm whether this dataset includes concept drift, we plotted the error rate per 300 adjacent messages by using a fixed Multi-Variate Bernoulli Naive Bayes [88], which was built from the first 1000 messages. As shown in Figure 3.7, its error rate fluctuated wildly, so classifier ensemble systems should not

[†3.2] Stop words are conjunctions, prepositions, articles, and other words such as 'and', 'a', 'of', 'for', and 'the' that appear often in messages yet alone may contain little meaning.

[†3.3] Stemming is a technique to reduce words to their grammatical roots so that they can be represented with an only term (e.g. 'learner' becomes 'learn').

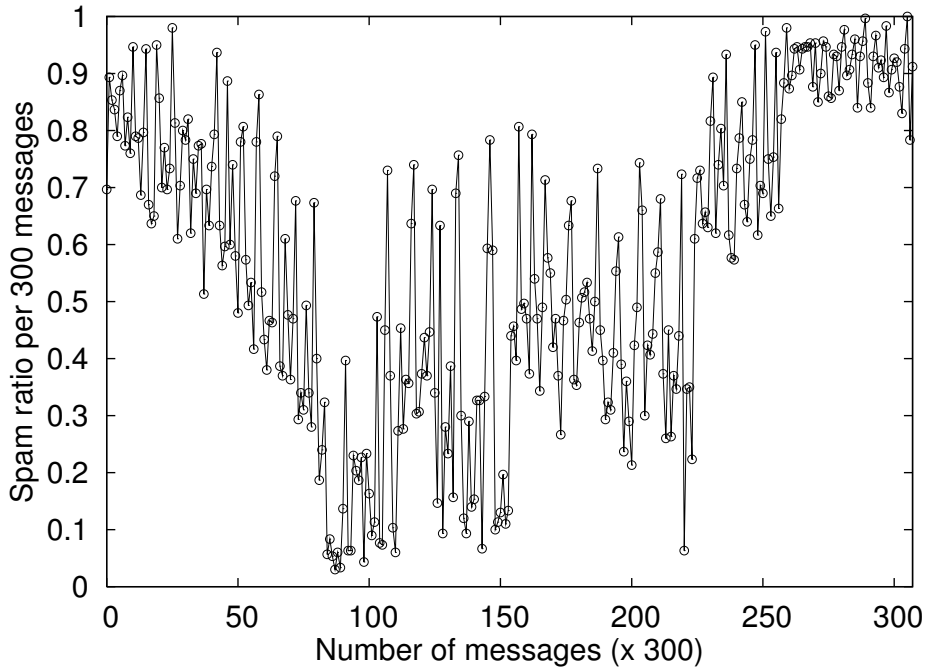

Fig. 3.6 Fluctuation of spam ratio on the TREC 2005 Spam Track Public Corpus (trec05p-1/full).

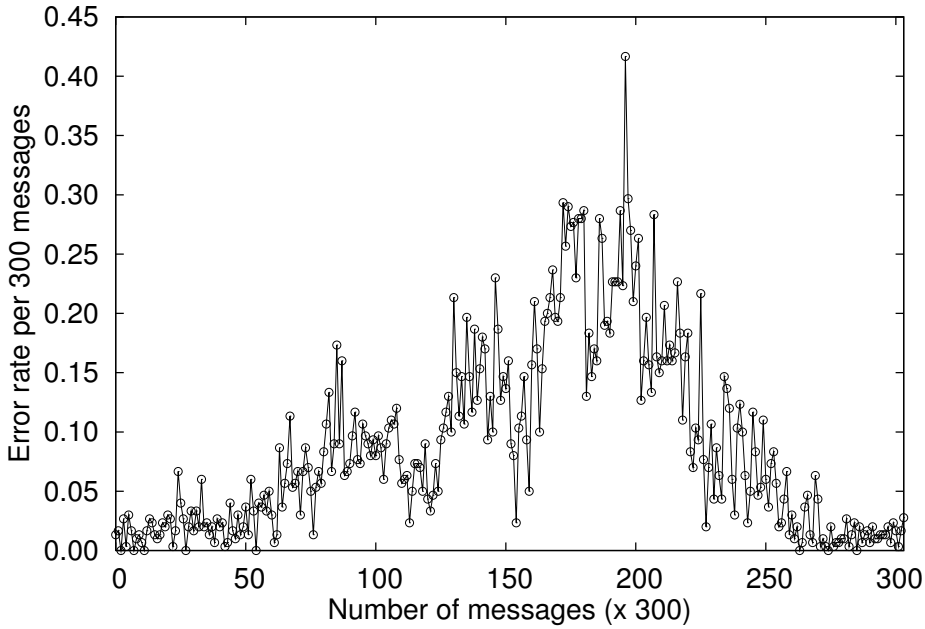

Fig. 3.7 Error rate of fixed Multi-Variate Bernoulli Naive Bayes built from the first 1000 messages in the TREC 2005 Spam Public Track Corpus (trec05p-1/full).

remove base classifiers when they performed poorly temporarily.

We first compared the enhanced ACE with the original ACE, AddExp, and WCEA to evaluate the overall effectiveness of the enhanced ACE as shown in Figure 3.8. The

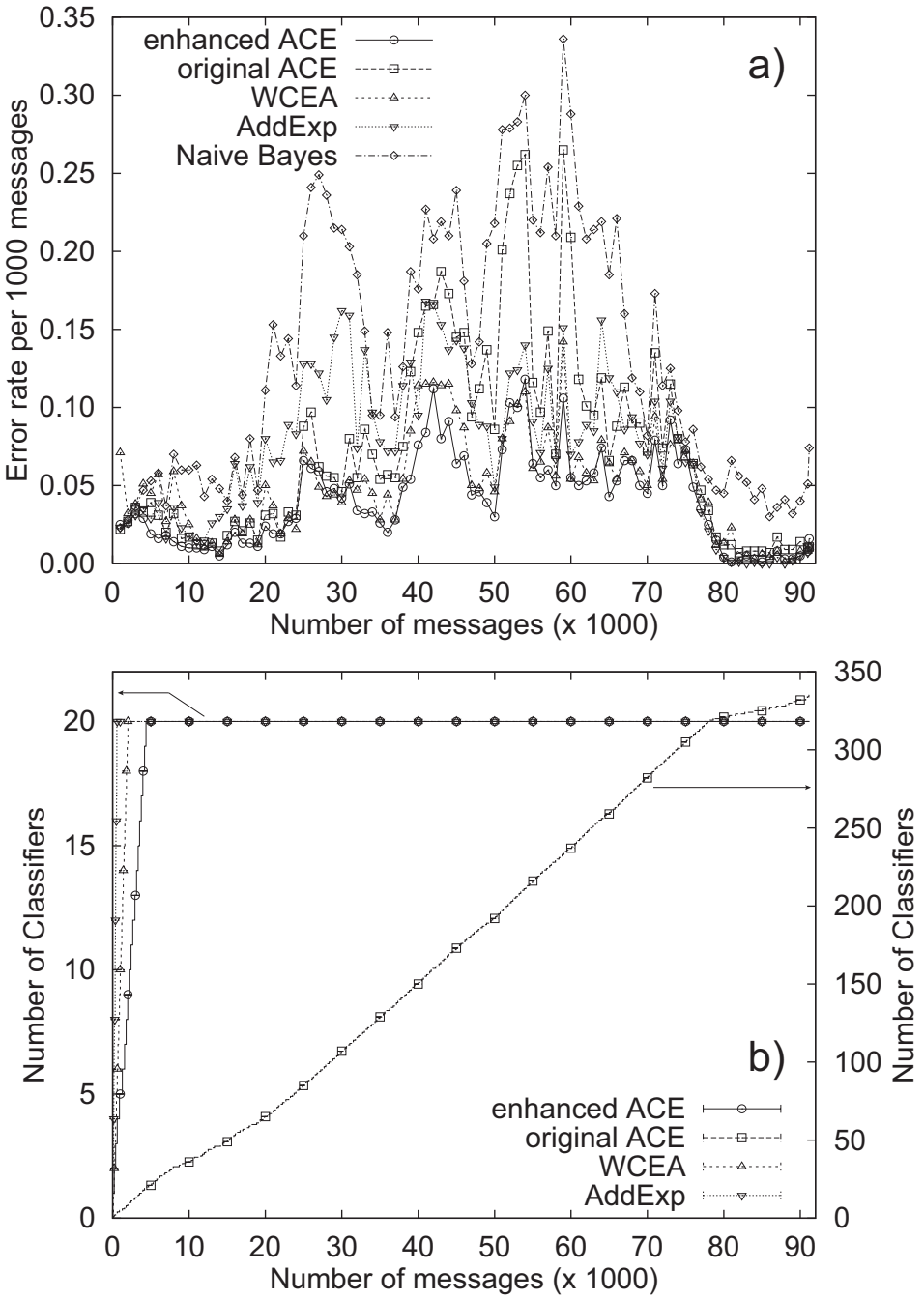

Fig. 3.8 (a) Error rates and (b) numbers of classifiers of the ACEs, WCEA, AddExp, and Multi-Variate Bernoulli Naive Bayes on the TREC 2005 Spam Track Public Corpus (trec05p-1).

parameter settings for the ACEs were $W = 100$, $L = 1200$, $\alpha = 0.9$, $\beta = 0.8$, and $M = 20$. The factor for decreasing weights and the factor for a new classifier weight for AddExp were 0.5 and 0.01. AddExp used the weakest first method. The chunk

size for WCEA was 200. The base classifiers for both the enhanced and original ACEs, AddExp, and WCEA were Multi-Variate Bernoulli Naive Bayes [88]. Each system, except the original ACE, had a capacity of 20 classifiers. Note that the original ACE does not remove any classifiers. All results were averaged over 500 trials.

Figure 3.8 shows that the enhanced ACE performed the best. Moreover, ACEs, WCEA, and AddExp performed better than the single Multi-Variate Bernoulli Naive Bayes. AddExp performed poorly at the middle period of the dataset. As we mentioned before, classifier ensemble systems should not remove their base classifiers when the classifiers perform poorly temporarily. AddExp was too sensitive to misclassifications of base classifiers and was thus unable to perform well. WCEA performed well over the whole dataset because it replaced classifiers with new ones more slowly than AddExp.

Also, the effectiveness of handling recurring concepts was low in this problem and the number of classifiers of the original ACE was over 300. Therefore, the interference from many old classifiers worsened the predictive accuracy of the original ACE significantly. Using the new pruning method and the improved weighting method enabled the enhanced ACE to perform better than the original ACE. This result clearly shows the importance of the new pruning method.

We then compared the enhanced ACE with the original ACE, "the original ACE with the new pruning method", and "the original ACE with the improved weighting method" to evaluate the effectiveness of the new features in the enhanced ACE. Table 3.1 shows their error rates and those of other systems. We can see that the new features enabled the original ACE to improve its predictive accuracy significantly. In particular, the improved weighting method enabled the original ACE to sufficiently reduce interference from old classifiers despite the fact that it had many classifiers.

Table. 3.1 Error rates of the ACEs, WCEA, AddExp, and Multi-Variate Bernoulli Naive Bayes on the TREC 2005 Spam Track Public Corpus (trec05p-1).

System	Error rate
Original ACE	0.0741
Original ACE with new pruning method	0.0432
Original ACE with improved weighting method	0.0432
Enhanced ACE	0.0404
WCEA	0.0498
AddExp	0.0724
NB	0.134
NB(1000)	0.107

Notes: NB is Multi-variate Bernoulli Naive Bayes[88], and NB(1000) is NB reinitialized every 1000 messages.

3.3.4 Short Summary

We have added a new classifier pruning method and we have improved the original weighting method to solve the drawbacks that the original ACE system has. The improved weighting method, which uses a confidence interval for the recent predictive accuracies of base classifiers, enables the enhanced ACE system to sufficiently reduce interference from old classifiers and thus to respond to sudden changes more quickly. And then, the new pruning method, which considers the occurrence of recurring concepts, enables the enhanced ACE system to remove redundant classifiers well. Experimental results confirmed the effectiveness of these new features in the enhanced ACE system.

3.4 Summary

Several systems have been proposed that use ensembles of classifiers for learning concept drift. Most of the systems build an ensemble of classifiers from sequential data streams and use a weighted majority vote from the outputs of up-to-date classifiers

instead of continuously revising a single classifier.

We proposed such an online learning system, which we call the adaptive classifiers-ensemble (ACE) system. ACE consists of a single online classifier, many batch classifiers, and a drift detection mechanism that uses a confidence interval for the recent predictive accuracies of the batch classifiers. The weighting method of ACE, which combines the outputs of base classifiers, uses the predictive accuracy of each classifier for recent training examples in order to avoid interference between the outputs of classifiers from old concepts and classifiers for new ones. ACE focuses especially on handling recurring concepts, and experimental results showed that ACE was better able to handle recurring concepts than other conventional classifier ensemble systems. However, the original ACE system does not have any classifier pruning method, and the original weighting method does not sufficiently reduce interference from old classifiers.

Therefore we have added a new classifier pruning method and we have improved the original weighting method. To handle recurring concepts in the pruning method, the enhanced ACE uses two sets of weights: one to decide the system output and the other one to determine the order of pruning. The improved weighting method uses a confidence interval for the recent predictive accuracies of base classifiers and thus sufficiently reduces the interference from old classifiers. Experimental results showed that the enhanced ACE system performed better than the original ACE system and conventional systems in learning concept drift.

Using both online and batch classifiers and a drift detection mechanism is a new approach to learning concept drift, and it enables ACE to respond to both sudden and gradual changes and to handle recurring concepts well.

In our future work, the improved weighting method and the new pruning method will be further improved. The pruning method especially is an ad-hoc method at present, so we need theoretical studies from now on. Then, the detection mechanism must also be improved. ACE could not detect concept drift accurately in real-world problems because the parameter settings for ACE are too difficult to use practically.

Therefore, we have proposed simpler systems of more accurately detecting concept drift, as presented in Chapters 4 and 5.

Chapter 4

Learning and Detecting Concept Drift with Two Online Classifiers

Detecting concept drift is important for dealing with real-world online learning problems. To detect concept drift in a small number of examples, several methods have been developed that monitor prediction errors of a single online classifier, as presented in Section 4.1. When these methods detect concept drift, the online classifier is reinitialized to prepare for the learning of the next concept. We have proposed a method of more accurately detecting concept drift. We describe this STEPD method that uses a *s*tatistical *t*est of *e*qual *p*roportions to *d*etect concept drift in Section 4.2. Results show that STEPD detected various types of concept drift quickly and accurately, but its false alarms degrade the predictive accuracy because STEPD uses only one online classifier. To reduce the bad influence of false alarms on predictive accuracy, we have proposed a system that uses two online classifiers, which we call the *t*wo *o*nline classifiers for learning and *d*etect*i*ng concept drift (Todi) system. We describe this Todi system in Section 4.3. Using two online classifiers also enables Todi to accurately notify a user of the occurrence of concept drift. We demonstrate that Todi performed well with an actual spam filtering dataset. Finally, we summarize this study in Section 4.4.

4.1 Background and Related Work

The detection of changes is one way to respond to concept drift. Examples of real problems for which change detection is relevant include user modeling, monitoring in biomedicine and industrial processes, fault detection and diagnosis, the safety of complex systems, etc. [5, 31]. There has been much work on detecting changes in online data streams [8, 12, 14, 46, 63, 64], but most of it is based on estimating the underlying distribution of examples, which requires a large number of examples.

To detect concept drift in a small number of examples, several methods have recently been proposed that monitor the prediction errors of a single online classifier [3, 31, 72, 75]. These methods assume that a significant increase in the error rate for the online classifier suggests that the target concept is changing. Also, these methods do not depend on the attributes of the type of input in contrast to methods that estimate the underlying distribution of examples.

This section describes related work, which is the drift detection method (DDM) [31], the early drift detection method (EDDM) [3], and ACE's drift detection method (ACED) [72, 75]. We present their pseudo-codes in Algorithms 4.1, 4.2, and 4.3.

4.1.1 Drift Detection Method

Gama et al. proposed a drift detection method using a single online classifier (DDM) [31]. For each time t, the error rate of the online classifier is the probability of misclassifying, p_t, with standard deviation, $s_t = \sqrt{p_t(1-p_t)/t}$. They assumed that p_t decreases as time advances if the target concept is stationary, and a significant increase in p_t suggests that the concept is changing. If the concept is unchanged, then the $100(1-\alpha)\%$ confidence interval for p_t with $n > 30$ examples is approximately $p_t \pm z_{\alpha/2} s_t$, where $z_{\alpha/2}$ denotes the upper $(100\alpha/2)$th percentile of the standard nor-

Algorithm 4.1 Drift detection method (DDM).

Online Classifier: $H : X \rightarrow Y$.
Initialize n, r, $p_{\min}$, $s_{\min} \leftarrow 0$, $\boldsymbol{B} \leftarrow \Phi$.
for each example ($\boldsymbol{x}_t \in X$, $y_t \in Y$) **do**
  Output: $H(\boldsymbol{x}_t)$.
  increment n. // number of observations.
  if $[\![H(\boldsymbol{x}_t) \neq y_t]\!]$ is **true then** // output was wrong.
    increment r. // number of wrong classifications.
  end if
  set $p_t \leftarrow r/n$. // probability of misclassifying.
  set $s_t \leftarrow \sqrt{p_t(1-p_t)/n}$. // standard deviation.
  if $p_t + s_t < p_{\min} + s_{\min}$ **then** // update registers.
    set $p_{\min} \leftarrow p_t$; $s_{\min} \leftarrow s_t$.
  end if
  train online classifier H with $(\boldsymbol{x}_t, y_t)$.
  if $n \geq 30$ **then**
    if $p_t + s_t < p_{\min} + 3s_{\min}$ **then** // concept drift occurred.
      rebuild online classifier H from $\boldsymbol{B}$.
      reinitialize n, r, $p_{\min}$, $s_{\min}$, $\boldsymbol{B}$.
    else if $p_t + s_t < p_{\min} + 2s_{\min}$ **then** // warning level.
      add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$. // short-term buffer.
    else // normal status.
      reinitialize $\boldsymbol{B}$.
    end if
  end if
end for

mal distribution. DDM stores the values of p_t and s_t when $p_t + s_t$ reaches its minimum value (obtaining $p_{\min}$ and $s_{\min}$) and stores examples in short-term memory while $p_t + s_t \geq p_{\min} + 2s_{\min}$ is satisfied. DDM then rebuilds the online classifier from the stored examples and reinitializes all variables if $p_t + s_t \geq p_{\min} + 3s_{\min}$ is reached. DDM performs well for sudden and significant changes, but DDM has difficulty detecting gradual and small changes because of the insensitivity of p_t to these changes. This insensitivity also causes the misdetection of sudden changes.

4.1.2 Early Drift Detection Method

To improve the detection of gradual changes, Baena-García et al. developed the early

Algorithm 4.2 Early drift detection method (EDDM).

Paramters: α_d, α_w: significant levels.
Online Classifier: $H : X \rightarrow Y$.
Initialize $n,\ r,\ p,\ q,\ s,\ p_{\max},\ s_{\max} \leftarrow 0$, $\boldsymbol{B} \leftarrow \Phi$.
for each example $(\boldsymbol{x}_t \in X,\ y_t \in Y)$ **do**
  Output: $H(\boldsymbol{x}_t)$.
  increment n. // number of observations.
  if $[\![H(\boldsymbol{x}_t) \neq y_t]\!]$ is **true then** // output was wrong.
    increment r. // number of wrong classifications.
    set $p \leftarrow ((r-1) * p + n)/r$. // average distance between two errors.
    set $q \leftarrow ((r-1) * q) + n^2)/r$; set $s \leftarrow \sqrt{q - p^2}$. // standard deviation.
    reinitialize n.
  end if
  if $p + 2s > p_{\max} + 2s_{\max}$ **then** // update registers.
    set $p_{\max} \leftarrow p$; $s_{\max} \leftarrow s$.
  end if
  train online classifier H with $(\boldsymbol{x}_t, y_t)$.
  if $r \geq 30$ **then** // 30 errors have happened.
    set $v \leftarrow (p + 2s)/(p_{\max} + 2s_{\max})$.
    if $v < \alpha_d$ **then** // concept drift occurred.
      rebuild online classifier H from $\boldsymbol{B}$.
      reinitialize n, r, p, q, s, $p_{\max}$, $s_{\max}$, $\boldsymbol{B}$.
    else if $v < \alpha_w$ **then** // warning level.
      add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$. // short-term buffer.
    else // normal status.
      reinitialize $\boldsymbol{B}$.
    end if
  end if
end for

drift detection method (EDDM) [3]. The key idea is to consider distances (time intervals) between two occurrences of classification errors. They assume that a significant decrease in distance suggests that the concept is changing. Thus, EDDM calculates the average distance between two errors, p_t, with standard deviation, s_t, and stores the values of p_t and s_t when $p_t + 2s_t$ reaches its maximum value (obtaining $p_{\max}$ and $s_{\max}$). EDDM also stores examples in short-term memory with $v_t < \alpha_w$ being satisfied, where $v_t = (p_t + 2s_t)/(p_{\max} + 2s_{\max})$. If $v_t < \alpha_d$ is reached, EDDM rebuilds the online classifier from the stored examples and reinitializes all variables. Note that EDDM starts detecting drift after 30 errors have occurred. EDDM performs well for gradual

Algorithm 4.3 ACE's drift detection method (ACED).

Parameters: W: window size, $1-\alpha$: significant level.
Online Classifier: $H : X \to Y$.
Initialize $r \leftarrow 0$, $\{w_t\} \leftarrow \Phi$.
for each example ($\boldsymbol{x}_t \in X$, $y_t \in Y$) **do**
  Output: $H(\boldsymbol{x}_t)$.
  increment n. // number of observations.
  set $w_t \leftarrow [\![H(\boldsymbol{x}_t) = y_t]\!]$.
  if w_t is **true then** // output was correct.
    increment r. // number of correct classifications among recent W examples.
  end if
  if w_{t-W} is **true then** // window overflowed.
    decrement r.
  end if
  train online classifier H with $(\boldsymbol{x}_t, y_t)$.
  $A_t \leftarrow (r+1)/(W+2)$. // recent accuracy.
  $A_t^l \leftarrow \frac{W}{W+z_{\alpha/2}^2}\left(A_t + \frac{z_{\alpha/2}^2}{2W} - z_{\alpha/2}\sqrt{\frac{A_t(1-A_t)}{W} + \frac{z_{\alpha/2}^2}{4W^2}}\right)$. // lower endpoint of CI.
  if $n \geq 2W$ and $A_t < A_{t-W}^l$ **then** // concept drift occurred.
    reinitialize online classifier H; reinitialize r.
  end if
end for

changes because the average distance between two errors is more sensitive to changes than the probability of misclassifying with DDM. However, the high sensitivity also causes many false alarms in noisy environments.

4.1.3 ACE's Drift Detection Method

The online learning system with multiple classifiers that we previously mentioned, ACE, also uses a drift detection method [72, 75]. We have simplified the method in order to detect concept drift with a single online classifier [73]. This simplified method, ACED, observes the predictive accuracy of the online classifier for the most recent W examples, A_t, and calculates the $100(1-\alpha_d)\%$ confidence interval for A_t at each time t. The key idea is that A_t will not fall below A_{t-W}^l if the target concept is stationary, where A_{t-W}^l is the lower endpoint of the interval at time $t-W$. ACED initializes

the classifier if $A_t < A^l_{t-W}$ is reached. Note that ACED starts detecting drift after receiving $2W$ examples. ACED is able to detect concept drift quickly when W is small, but such small windows often cause false alarms.

4.2 The STEPD method

To achieve a more accurate detection of concept drift, we have proposed the STEPD method that monitors predictive accuracy of a single online classifier [73]. In this section, we first describe the detection of concept drift with a statistical test of equal proportions. We then demonstrate that STEPD was able to detect concept drift quickly and accurately in five synthetic datasets that contain various types of concept drift. Finally, we briefly summarize the study of STEPD. Algorithm 4.4 shows the pseudo-code of STEPD.

4.2.1 Detection with Statistical Test of Equal Proportions

The basic principle to consider is the two predictive accuracies of the online classifier: the recent one and the overall one. We assume that there is a statistical significance, after the occurrence of concept drift, between the accuracy for recent examples and the overall accuracy from the beginning of the learning. STEPD compares these two accuracies (populations of correct classifications) by using a statistical test of equal proportions.

Let n be the number of examples that the online classifier learned, s be the number of correct classifications among the most recent W examples, and r be the number of correct classifications among $n - W$ examples (excepting the W examples from the n examples). Then, let $\hat{p}_A = r_A/n_A = r/(n - W)$ and $\hat{p}_B = r_B/n_B = s/W$ be the two proportions. The statistical test of equal proportions that STEPD uses is performed

Algorithm 4.4 STEPD method.

Parameters: W: window size, $1-\alpha_d$, $1-\alpha_w$: significant levels.
Online Classifier: $H : X \to Y$.
Initialize n, r, $s \leftarrow 0$, $\{w_t\}$, $\boldsymbol{B} \leftarrow \Phi$.
for each example ($\boldsymbol{x}_t \in X$, $y_t \in Y$) **do**
  Output: $H(\boldsymbol{x}_t)$.
  // updating classification results. ////////////////////
  increment n.
  set $w_t \leftarrow [\![H(\boldsymbol{x}_t) = y_t]\!]$.
  if w_t is **true then** // output was correct.
    increment s.
  end if
  if w_{t-W} is **true then** // window overflows.
    increment r; decrement s.
  end if
  train online classifier H with $(\boldsymbol{x}_t, y_t)$.
  // detecting concept drift. ////////////////////
  if $n \geq 2W$ **then**
    $P \leftarrow$ P-value of $T(r, s, n-W, W)$.
    if $r/(n-W) > s/W$ and $P < \alpha_d$ **then** // concept drift occurred.
      rebuild the classifier from $\boldsymbol{B}$.
      reinitialize n, r, s, $\{w_t\}$, $\boldsymbol{B}$.
    else if $r/(n-W) > s/W$ and $P < \alpha_w$ **then** // warning level.
      add $(\boldsymbol{x}_t, y_t)$ to $\boldsymbol{B}$. // short-term buffer.
    else // normal status.
      reinitialize $\boldsymbol{B}$.
    end if
  end if
end for

by calculating the following statistic[†4.1],

$$T(r_A, r_B, n_A, n_B) = \frac{|r_A/n_A - r_B/n_B| - 0.5(1/n_A + 1/n_B)}{\sqrt{\hat{p}(1-\hat{p})(1/n_A + 1/n_B)}}, \tag{4.1}$$

where $\hat{p} = (r_A + r_B)/(n_A + n_B)$, and then by comparing this statistic value with the percentile of the standard normal distribution to obtain the P-value of the test (P). If $\hat{p}_A > \hat{p}_B$ and $P < \alpha$, where α is a one-tailed significance level, the null hypothesis that

[†4.1] We should use the Fisher's exact test where sample sizes are small [1, 27]. However, we did not use it due to its high computational costs. The statistic in Equation (4.1) is equivalent to Peason's chi-square test with Yates' correction for continuity [102].

the population proportions are equal ($p_A = p_B$; concept drift did not occur) is rejected and the alternative hypothesis that the population proportions are not equal ($p_A > p_B$; concept drift occurred) is accepted. To summarize, STEPD tests a significant decrease in the recent accuracy $\hat{p}_B = s/W$, which is caused by concept drift, compared with the overall accuracy $\hat{p}_A = r/(n - W)$.

STEPD also uses two significance levels, which are α_w (warning) and α_d (drift), as in the DDM and EDDM methods. STEPD thus stores examples in short-term memory while $r/(n - W) > s/W$ and $P < \alpha_w$ are satisfied, and then rebuilds the classifier from the stored examples and reinitializes all variables when $r/(n - W) > s/W$ and $P < \alpha_d$ are reached. Note that STEPD starts detecting drift after satisfying $n \geq 2W$ and the stored examples are removed if $P \geq \alpha_w$ is satisfied.

4.2.2 Experiments and Results

We used five synthetic datasets that contain various types of concept drift [3, 31, 96]. The datasets have two classes. Each concept has 1000 examples. The total number of training examples is 4000, except for STAGGER in which the number is 3000. At each time step, an online learning system was trained with one example and then was tested with 100 examples generated randomly according to the current concept.

- STAGGER (1S). sudden. The dataset has three nominal attributes: *size* (small, medium, large), *color* (red, blue, green), and *shape* (circle, square, triangle), and has three concepts: 1) [*size* = small and *color* = red], 2) [*color* = green or *shape* = circle], and 3) [*size* = medium or large].
- GAUSS (2G). sudden, noisy. The examples are labeled according to two different but overlapped Gaussian, $N((0, 0), 1])$ and $N((2, 0), 4)$. The overlapping can be considered as noise. After each change, the classification is reversed.
- MIXED2 (3M). sudden, noisy. The dataset has two Boolean attributes (v, w) and two continuous attributes (x, y) from $[0, 1]$. The examples are classified

as positive if at least two of the three following conditions are satisfied: v, w, $y < 0.5 + 0.3\sin(3\pi x)$. After each change, the classification is reversed. Noise is introduced by switching the labels of 10% of the examples.

- CIRCLES (4C). gradual. The examples are labeled according to the condition: if an example is inside the circle, then its label is positive. The change is achieved by displacing the center of the circle ($(0.2, 0.5) \rightarrow (0.4, 0.5) \rightarrow (0.6, 0.5) \rightarrow (0.8, 0.5)$) and growing its radius ($0.15 \rightarrow 0.2 \rightarrow 0.25 \rightarrow 0.3$).
- HYPERP (5H). quite gradual. The examples uniformly distributed in multidimensional space $[0, 1]^{10}$ are labeled satisfying $\sum_{i=1}^{10} a_i x_i \geq a_0$ as positive. The weights of the moving hyperplane, $\{a_i\}$, which are initialized to $[-1, 1]$ randomly, are updated as $a_i \leftarrow a_i + 0.001 s_i$ at each time, where $s_i \in \{-1, 1\}$ is the direction of change for each weight. The threshold a_0 is calculated as $a_0 = \frac{1}{2}\sum_{i=1}^{10} a_i$ at each time. $\{s_i\}$ is reset randomly every 1000 examples.

We compared STEPD with DDM, EDDM, ACED, and classifiers that did not use any detection methods (Not Use). The parameter settings for STEPD and ACED were $W = 30$, $\alpha_d = 0.003$, and $\alpha_w = 0.05$. Those for EDDM were $\alpha = 0.95$ and $\beta = 0.90$. We used two distinct classifiers, the Weka implementations of IB1 (1-Nearest Neighbor) and of Naive Bayes (NB) [2, 44, 101], with these detection methods. All results were averaged over 500 trials.

Figure 4.1 and Table 4.1 show that all the detection methods improved the performance of the two classifiers in all the datasets. STEPD performed the best for sudden changes (1S, 2G, and 3M). Moreover, STEPD was comparable to EDDM for gradual changes (4C and 5H). ACED and EDDM were able to detect gradual changes well, but they gave many false alarms because they were too sensitive to errors and noise (see their numbers of detection (N_d) for 2G and 3M in Table 4.1). DDM detected sudden changes correctly, but its detection speed was quite slow. STEPD performed well for both sudden and gradual changes.

Table. 4.1 Cumulative prediction error rates with 95% confidence interval, numbers of detection (N_d), and numbers of required examples to detect drift (N_e).

Data set	Method	IB1			Naive Bayes (NB)		
		Error Rate	N_d	N_e	Error Rate	N_d	N_e
1S	STEPD	**.0059** ± .0001	**2.000** – **.000**	**4.29**	**.0076** ± .0002	1.998 – .010	**4.89**
	DDM	.0064 ± .0001	**2.000** – .106	7.35	.0087 ± .0002	**2.000** – .370	8.94
	EDDM	.0214 ± .0000	**2.000** – **.000**	47.3	.0208 ± .0000	**2.000** – **.000**	42.0
	ACED	.0085 ± .0001	**2.000** – **.000**	11.4	.0100 ± .0002	**2.000** – .008	11.4
	Not Use	.3134 ± .0007			.3351 ± .0006		
2G	STEPD	**.1676** ± .0007	**2.966** – 1.102	**10.6**	**.1109** ± .0004	2.964 – 1.180	**7.89**
	DDM	.2039 ± .0044	2.766 – **.892**	35.2	.1347 ± .0029	**2.970** – **1.008**	26.0
	EDDM	.1898 ± .0016	2.918 – 10.56	26.4	.1250 ± .0009	2.934 – 7.682	18.0
	ACED	.1749 ± .0008	2.880 – 5.306	12.0	.1156 ± .0005	2.910 – 3.434	9.68
	Not Use	.4456 ± .0006			.4737 ± .0007		
3M	STEPD	**.2143** ± .0009	**2.968** – .932	**12.8**	**.1885** ± .0006	**2.976** – .586	**11.2**
	DDM	.2439 ± .0036	2.672 – **.748**	43.9	.2008 ± .0013	2.942 – **.364**	36.8
	EDDM	.2443 ± .0014	2.884 – 13.20	33.6	.2175 ± .0009	2.952 – 8.704	33.5
	ACED	.2262 ± .0009	2.866 – 6.690	14.1	.2043 ± .0008	2.850 – 5.604	12.8
	Not Use	.4534 ± .0007			.4864 ± .0007		
4C	STEPD	**.0286** ± .0002	**2.952** – **.190**	**26.8**	.0956 ± .0007	1.584 – **2.292**	42.5
	DDM	.0320 ± .0003	2.318 – 1.490	58.9	.1072 ± .0010	.686 – 3.450	60.9
	EDDM	.0318 ± .0002	2.618 – .462	49.5	**.0920** ± .0004	**1.588** – 7.934	50.0
	ACED	.0529 ± .0002	1.498 – .908	31.6	.1046 ± .0009	.786 – 2.952	**37.5**
	Not Use	.1365 ± .0004			.1536 ± .0005		
5H	STEPD	**.2254** ± .0012	1.406		.1182 ± .0014	2.000	
	DDM	.2361 ± .0016	.048		.1278 ± .0017	1.518	
	EDDM	.2327 ± .0013	6.834		**.1110** ± .0011	4.800	
	ACED	.2326 ± .0009	7.486		.1176 ± .0011	3.398	
	Not Use	.2465 ± .0021			.1590 ± .0028		

Notes: The prediction error rate is only calculated from the error on training data. The form of the N_d column ($n-m$) means that n is the number of detections within 100 examples after each change and m corresponds to the number of false alarms. We excluded trials where false alarms occured from the calculation of N_e.

4.2.3 Short Summary

Our proposed method, STEPD, uses a statistical test of equal proportions to detect concept drift. Experiments showed the test enabled STEPD to detect various types of concept drift quickly and accurately.

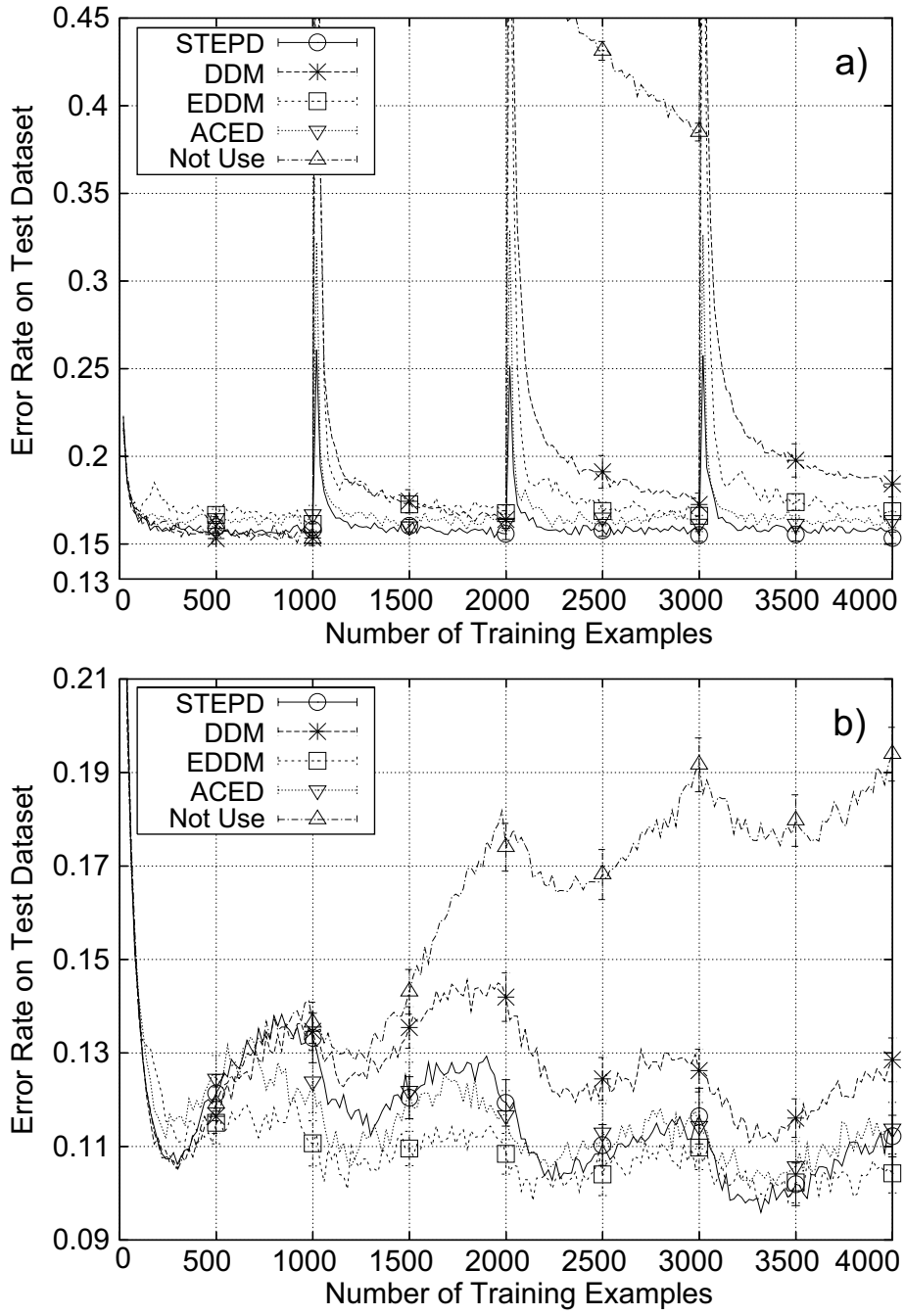

Fig. 4.1 Test error rates with 95% confidence intervals of STEPD, DDM, EDDM, ACED, and IB1 or Naive Bayes that did not use any detection methods. (a) 2G–IB1. (b) 5H–NB.

DDM, EDDM, and STEPD store training examples in short-term memory while a warning level is reached and then rebuild the online classifier from the stored examples if a drift level is reached. Although this memory slightly improves their predictive

accuracy immediately after the rebuild of the online classifier, their false alarms worsen their predictive accuracies significantly because these methods only use a single online classifier. We therefore have developed a system that uses two online classifiers, the Todi system. We describe the Todi system in the next section.

4.3 The Todi System

To reduce the bad influence of false alarms on predictive accuracy, we have proposed the *t*wo *o*nline classifiers for learning and *d*etect*i*ng concept drift (Todi) system. Todi uses two online classifiers: one that is reinitialized and the other one that is not reinitialized when concept drift is detected. In this section, we first explain how the two classifiers are trained and how the system output is selected from the outputs of the two classifiers. Next, we describe the detection method that uses the statistical test of equal proportions and discuss confirming the correctness of detection. We then demonstrate that our Todi system was able to learn and detect concept drift quickly and accurately in both synthetic and real datasets. Finally, we briefly summarize the study of the Todi system. Algorithm 4.5 shows the pseudo-code of Todi.

4.3.1 Training and Selecting Classifiers

Todi trains both two online classifiers, H_0 and H_1, with a new example at each time t, and then stores their classification results, $w_{0,t}$ and $w_{1,t}$, in sliding windows for the most recent W examples. Todi manages five counters: n, r_0, r_1, s_0, and s_1, where n is the number of examples that H_0 learned; s_i is the number of correct classifications of H_i among the W examples; and r_i is the number of correct classifications of H_i among $n - W$ examples (excepting the W examples from the n examples).

Next, to decide the system output, Todi selects either output among the outputs of the two classifiers using the values of s_0 and s_1. If $s_0 > s_1 + 1$ then Todi selects

Algorithm 4.5 Todi system.

Parameters: W: window size, $1-\alpha$, $1-\beta$: significant levels.
Online Classifiers: $H_0, H_1 : X \to Y$.
Initialize $n, r_0, r_1, s_0, s_1 \leftarrow 0$, $\{w_{0,t}\}, \{w_{1,t}\} \leftarrow \Phi$, $c \leftarrow 0$.
for each example $(\boldsymbol{x}_t \in X, y_t \in Y)$ **do**
  Output: $H_c(\boldsymbol{x}_t)$. // the output of selected classifier ($c = 0$ or 1).
  // updating classification results. ////////////////////
  increment n.
  for each classifier H_i ($i = 0$, 1) **do**
    get classifier output $H_i(\boldsymbol{x}_t) \in Y$.
    set $w_{i,t} \leftarrow [\![H_i(\boldsymbol{x}_t) = y_t]\!]$. // **true** or **false**.
    if $w_{i,t}$ is **true then** // output was correct.
      increment s_i.
    end if
    if $w_{i,t-W}$ is **true then** // window overflows.
      increment r_i; decrement s_i.
    end if
    train both online classifiers H_0 and H_1 with $(\boldsymbol{x}_t, y_t)$.
  end for
  // selecting a classifier. ////////////////////
  if $s_0 > s_1 + 1$ **then**
    set $c \leftarrow 0$. // select H_0.
  else if $s_0 < s_1 - 1$ **then**
    set $c \leftarrow 1$. // select H_1.
  end if // else keep c.
  // detecting concept drift. ////////////////////
  if $n \geq 2W$ **then**
    $P_d \leftarrow$ P-value of $T(r_0, s_0, n - W, W)$.
    if $r_0/(n - W) > s_0/W$ and $P_d < \alpha$ **then** // concept drift occurred.
      $P_c \leftarrow$ P-value of $T(r_0 + s_0, r_1 + s_1, n, n)$.
      if $(r_0 + s_0)/n > (r_1 + s_1)/n$ and $P_c < \beta$ **then** // last detection was correct.
        set $H_1 \leftarrow H_0$, $s_1 \leftarrow s_0$, $\{w_{1,t}\} \leftarrow \{w_{0,t}\}$. // copy classifier H_0 in H_1.
      end if// else keep H_1, s_1, $\{w_{1,t}\}$.
      reinitialize H_0, n, r_0, r_1, s_0, $\{w_{0,t}\}$.
    end if
  end if
end for

$H_0(x_t)$, else if $s_0 < s_1 - 1$ then Todi selects $H_1(x_t)$, and else Todi selects $H_c(x_t)$, where Todi selected $H_c(x_{t-1})$ at the last time $t-1$ ($c = 0$ or 1). Note that Todi selects H_0 at first and selects H_1 immediately after the occurrence of concept drift. This selection method prevents the system output from alternating between H_0 and H_1 too

frequently in order not to degrade the predictive accuracy of the system output.

4.3.2 Detecting Concept Drift

To detect concept drift, Todi uses the slightly modified STEPD method, which only uses one significant level, with the online classifier H_0. Let α be a one-tailed significance level, and Todi calculates the P-value of the statistic $T(r_0, s_0, n - W, W)$, P_d. If $r_0/(n - W) > s_0/W$ and $P_d < \alpha$ are reached, Todi assumes that concept drift occurred and then reinitializes H_0 to learn the next concept from scratch. Note that Todi starts detecting concept drift after satisfying $n \geq 2W$.

The original STEPD method uses two significant levels (warning and drift) and stores examples in short-term memory while the warning level is satisfied. The original STEPD method then rebuilds the online classifier from the stored examples when the drift level is reached. Although this memory slightly improves the predictive accuracy immediately after the detection of concept drift, false alarms degrade the predictive accuracy significantly. Moreover, STEPD has a trade-off between accurate detection and quick detection.

To overcome these problems, Todi adds the another online classifier, H_1, which is not reinitialized when concept drift is detected, instead of using two significant levels and storing examples in short-term memory. The predictive accuracy of Todi does not seriously deteriorate if false alarms occur, because Todi can use H_1 that keeps a high predictive accuracy immediately after the reinitialization of H_0. Todi is thus able to use parameters that realize quick detection without the fear of false alarms.

4.3.3 Notifying Users of Concept Drift

Using the another online classifier, H_1, also enables Todi to confirm the correctness of the last detection and to accurately notify a user of the occurrence of concept drift.

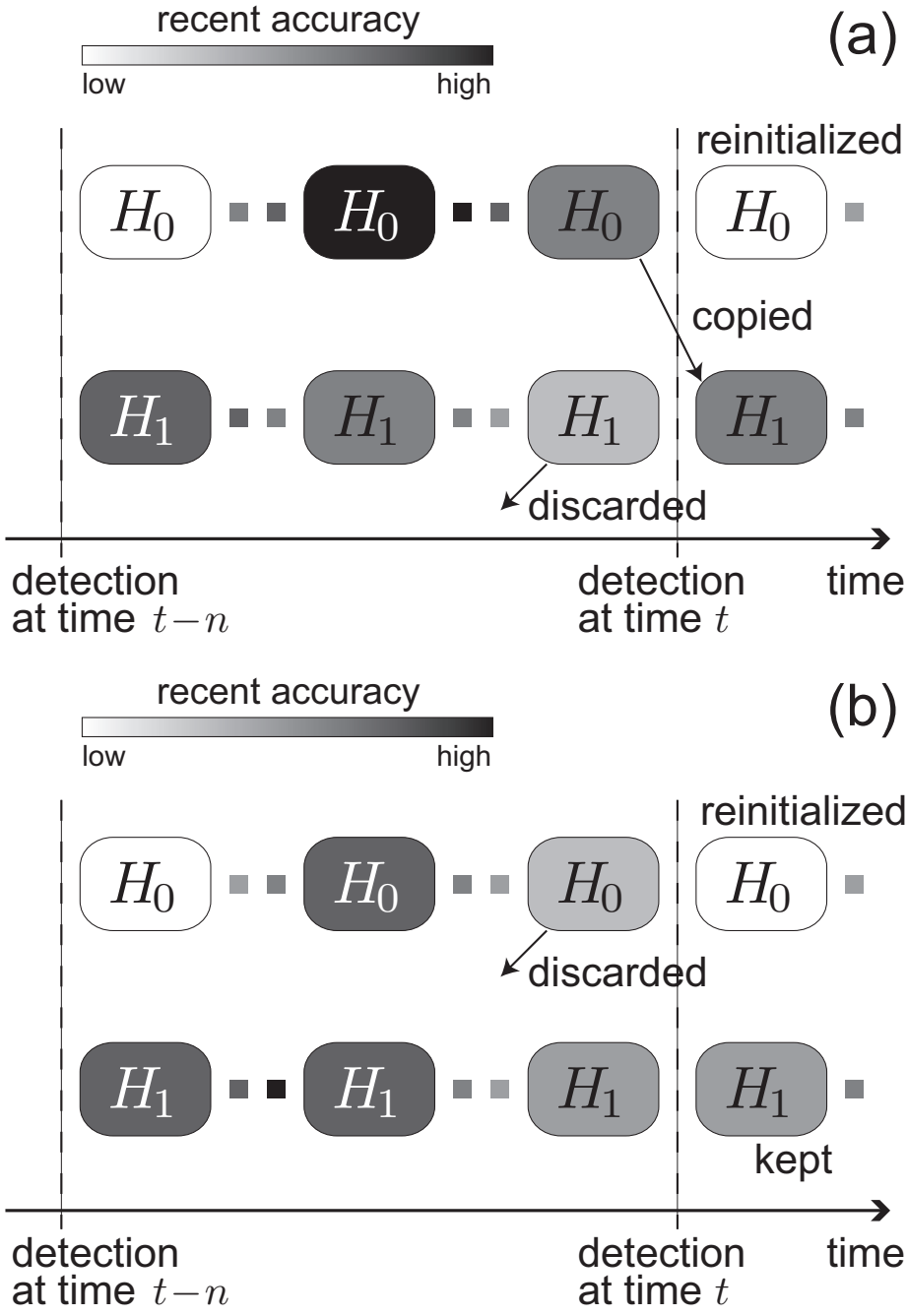

Fig. 4.2 Outline of detection procedure by Todi. (a) The procedure when the last detection at time $t-n$ was correct ($P_c < \beta$) and (b) the other procedure when the detection was wrong. Todi notifies a user of the occurrence of concept drift when $P_c < \beta$ is reached.

When concept drift is detected, Todi calculates the P-value of the statistic $T(r_0 + s_0, r_1 + s_1, n, n)$, P_c. Todi confirms that the last detection was correct if $(r_0 + s_0)/n >$

$(r_1 + s_1)/n$ and $P_c < \beta$ are satisfied, and then notifies a user of the occurrence of concept drift. The β is another one-tailed significance level. That is to say, when the last reinitialization of H_0 was effective in handling concept drift significantly, a user is notified of the occurrence of concept drift.

When $P_c < \beta$ is reached (the last detection was correct), Todi keeps H_0. Todi thus copies H_0 (including s_0 and $\{w_{0,t}\}$) in H_1 before the reinitialization of H_0. Note that n, r_0, r_1, s_0, and $\{w_{0,t}\}$ are reinitialized when $P_d < \alpha$ is reached. In contrast, when $P_c < \beta$ is not reached, Todi keeps H_1 and reinitializes H_0. Figure 4.2 shows an outline of the detection procedure by Todi.

4.3.4 Experiments and Results

We used two datasets in our experiments. The first one is a synthetic dataset, which is the SEA concepts [92], and the second one is an actual spam filtering dataset, which is the TREC 2006 Spam Track Public Corpora [16]. The results demonstrate that the Todi system is able to learn and detect concept drift quickly and accurately.

A SEA concepts

In the first experiment, we used the SEA concepts, which is a benchmark dataset for concept drift [49, 92], to evaluate the predictive accuracy of and the notification of Todi. This problem has three attributes, $\boldsymbol{x} \in [0, 10]^3$. The target concept is $y = [\![x_0 + x_1 \leq \theta]\!]$, namely each example belongs to class 1 if $x_0 + x_1 \leq \theta$ and class 0 otherwise. Thus, only the first two attributes are relevant.

Threshold values were $\theta = 8$, 9, 7, and 9.5 for four data blocks. Each block lasted 12500 time steps, and there was a period where θ was not deterministic for ΔT time steps at the beginning of each block. For instance, in the case where $\Delta T = 1000$, $\Pr(\theta = 9) = 1/1000$ and $\Pr(\theta = 8) = 999/1000$ at time 12501, On the other hand, in the case where $\Delta T = 1$, $\Pr(\theta = 9) = 1$ and $\Pr(\theta = 8) = 0$ at time 12501. We used

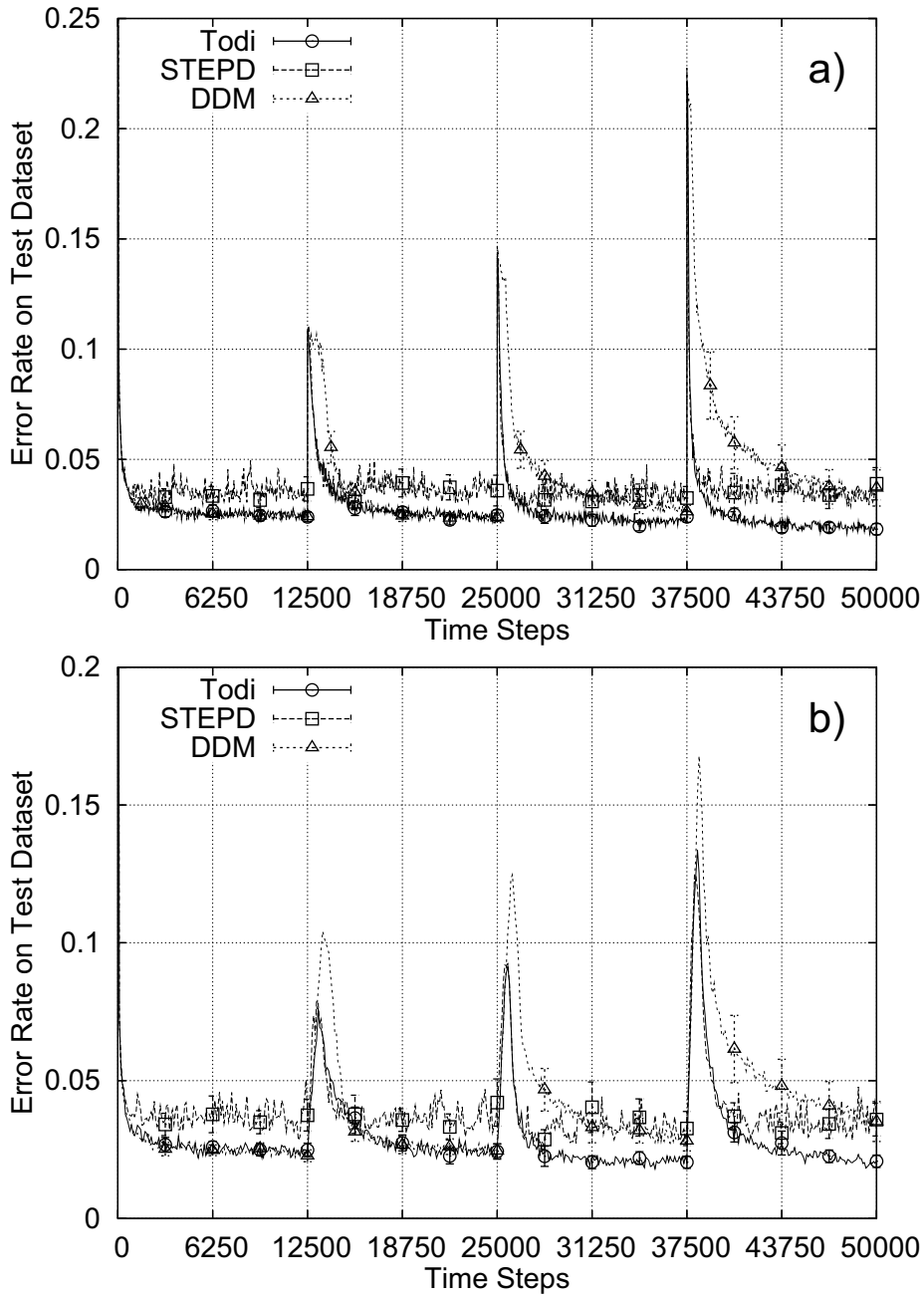

Fig. 4.3 Test error rates of Todi, STEPD, and DDM on the SEA concepts. (a) $\Delta T = 1$ and (b) $\Delta T = 1000$. Error bars show the standard errors from 100 trials.

$\Delta T = 1$ and 1000 ($\Delta T = 1$ is the original setting). There were three changes, and the detection of the first change was the most difficult because the difference between thresholds $\theta = 8$ and 9 were the smallest among the three changes. The total number of training examples was 50000. Noise was introduced by switching the labels of 10% of

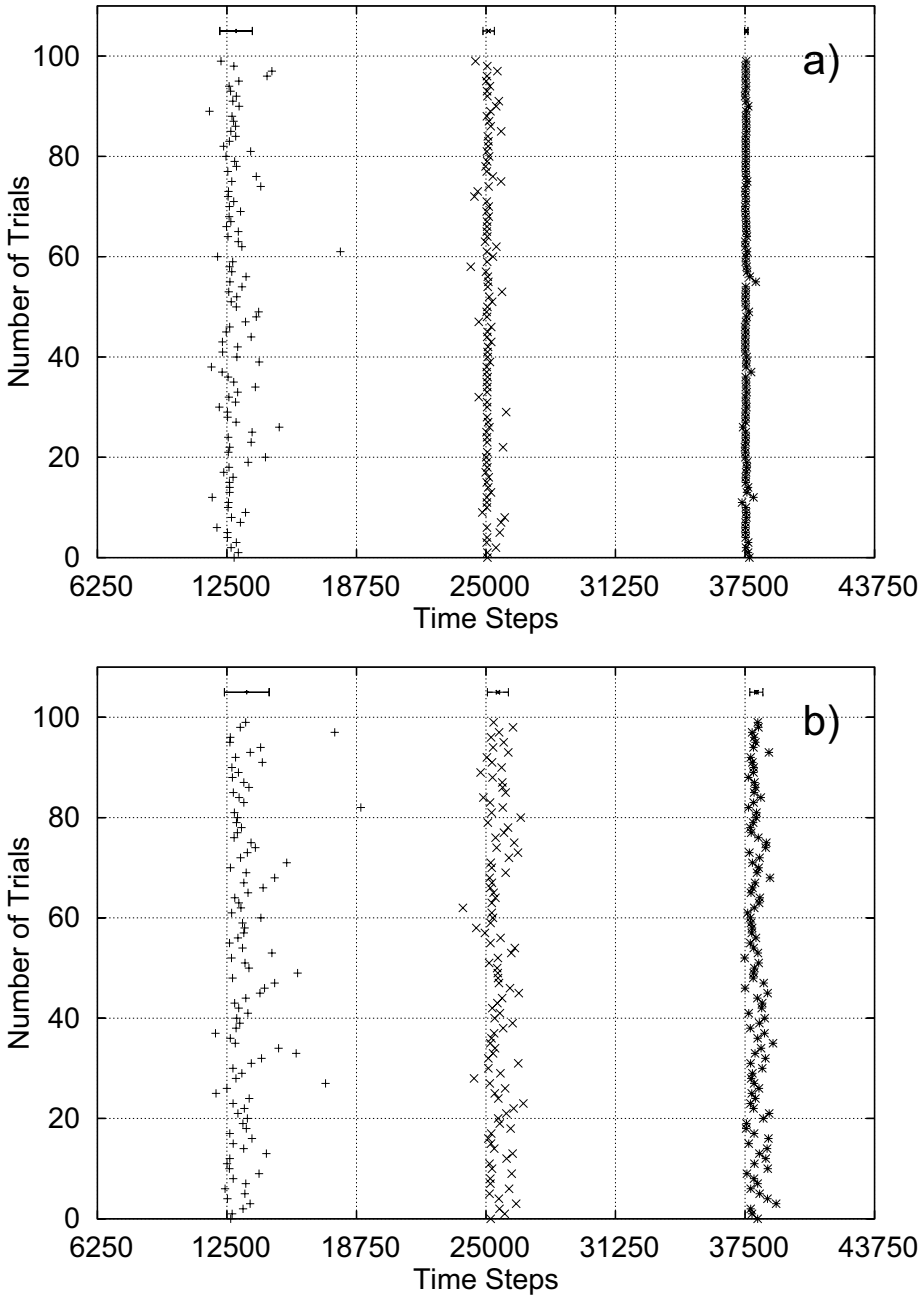

Fig. 4.4 Results that Todi notified a user of the occurrence of concept drift on the SEA concepts. (a) $\Delta T = 1$ and (b) $\Delta T = 1000$. These results were clustered by the Weka implementation of EM algorithm (`numClusters = 3`). Error bars show the standard deviations of each cluster from 100 trials (Each cluster had 100 results).

the training examples. At each time step, an online learning system was trained with one example and then was tested with 200 examples generated randomly according to the current concept.

We compared Todi with STEPD and DDM. The parameter settings for Todi were $W = 50$, $\alpha = 0.01$, and $\beta = 0.01$. These systems used Naive Bayes that can handle numeric attributes [44, Equations (1) – (3)] for their online classifiers. STEPD used a window of size 50 and two significant levels, 0.01 (drift) and 0.05 (warning). All results were averaged over 100 trials.

Figure 4.3 shows that STEPD gave many false alarms and thus its predictive accuracy deteriorated. Using two online classifiers enabled Todi to perform well in the presence of false alarms that are caused by noise. Todi performed the best for both sudden ($\Delta T = 1$) and gradual ($\Delta T = 1000$) changes. DDM gave much misdetection and its detection speed was slow.

Figure 4.4 shows the results that Todi notified a user of the occurrence of concept drift on the SEA concepts. There were no misdetection and were no false alarms among the notifications. Todi is able to detect significant changes that considerably degrade the predictive accuracy of an online classifier even if the target concept changes gradually.

B TREC 2006 Spam Track Public Corpora

In the second experiment, we evaluated the predictive accuracy of Todi with an actual dataset, the TREC 2006 Spam Track Public Corpora [16]. We used the public English corpus (trec06p/full) and conducted the immediate feedback filtering task where an online learning system is trained with one message at each time step. The dataset has 37,822 raw messages in chronological order of which about 65.9% are spam [15]. Spam filtering is problematic because the topics of and the characteristics of messages shift over time, and the spam ratio fluctuates wildly (see Figure 4.5).

We used Bogofilter [83], which is one of the well-known and successful spam filters, for online classifiers that Todi uses in this experiment. Bogofilter classifies a message as spam or ham (non-spam) by a statistical analysis of the message's header and content (body). The analysis is based on the use of a Bayesian technique that was

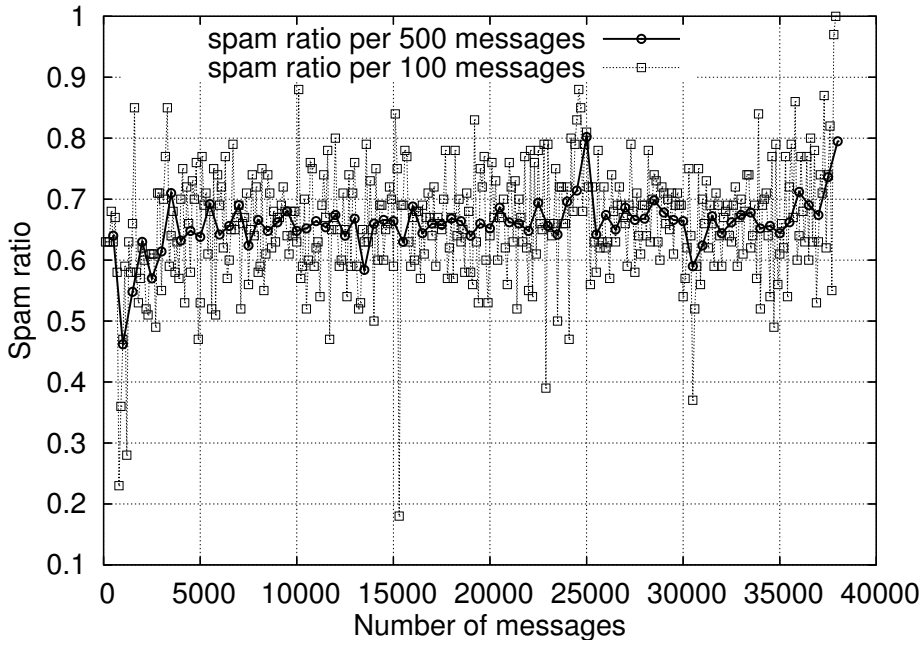

Fig. 4.5 Fluctuation of spam ratio on the TREC 2006 Spam Track Public Corpora (English corpus: trec06p/full).

Table. 4.2 Numbers and rates of false positive (FP), false negative (FN), and error (FP+FN) of Todi, STEPD, and Bogofilter on the TREC 2006 Spam Track Public Corpora (English corpus: trec06p/full, immediate feedback filtering task).

Systems	No. of FP	FP rate	No. of FN	FN rate	No. of Error	Error rate
Todi	11	8.52×10^{-4}	859	0.0345	870	0.0230
STEPD	62	4.80×10^{-3}	2170	0.0871	2232	0.0590
Bogofilter	7	5.42×10^{-4}	995	0.0399	1002	0.0265

Notes: False positive means the misclassification of ham messages and false negative means the misclassification of spam messages.

described by Paul Graham [32, 33] and some refinements that were described by Gary Robinson [84, 85].

We compared Todi with STEPD and Bogofilter. The parameter settings for Todi were $W = 50$, $\alpha = 0.05$, and $\beta = 0.05$. Todi and STEPD used Bogofilter for their online classifiers. STEPD used a window of size 50 and two significant levels, 0.05 (drift) and 0.20 (warning). Bogofilter used the default parameters, except for the spam-cutoff value that was set to 0.55.

We can see that the number of concept drift that Todi notified to a user was only one

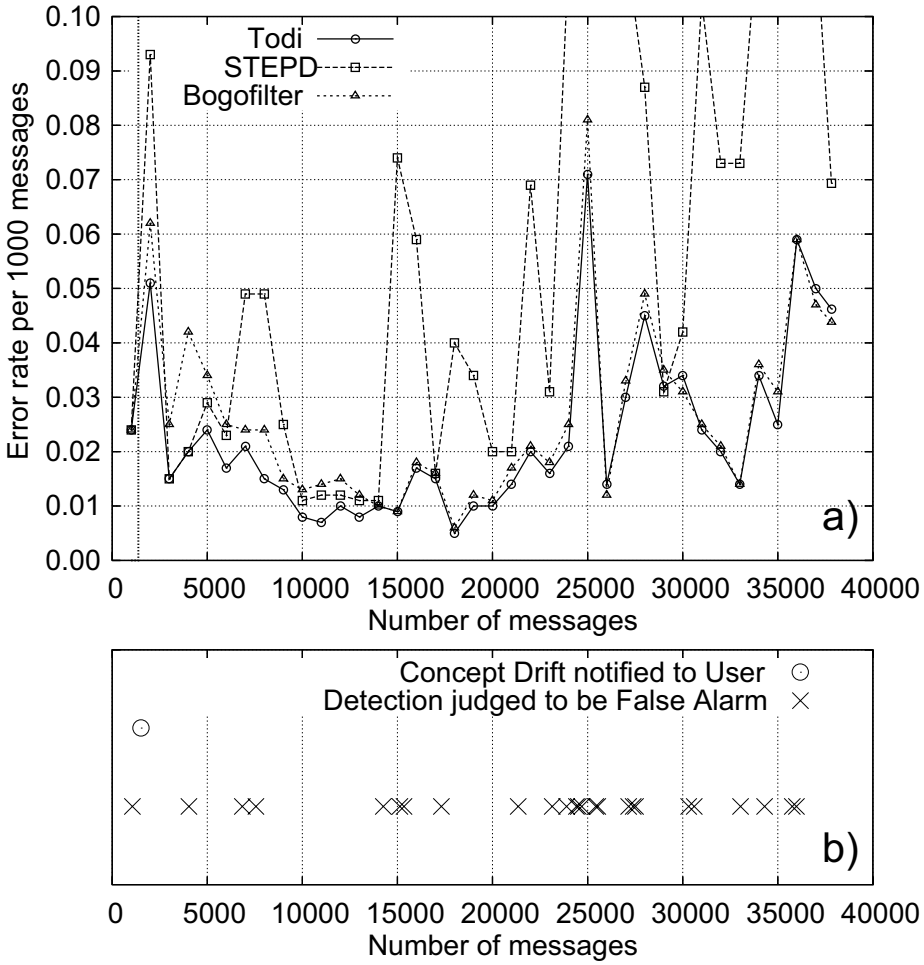

Fig. 4.6 (a) Error rates per 1000 messages of Todi, STEPD, and Bogofilter on the TREC 2006 Spam Track Public Corpora (English corpus: trec06p/full, immediate feedback filtering task). The vertical line at message 1526 indicates the concept drift that Todi notified to a user. (b) concept drift that Todi notified to a user and detection judged to be false alarm by Todi.

from Figure 4.6. Todi, with two Bogofilters, improved the predictive accuracy of the single Bogofilter after the notification. Moreover, there were periods that Bogofilter performed temporarily poorly because the spam ratio of this dataset fluctuated wildly as shown in Figure 4.5 and the topics of and the characteristics of messages shifted over time. Such periods caused the false alarms of STEPD as shown in Figure 4.6 (b) with the false alarms worsening the predictive accuracy significantly. However, Todi reduces the bad influence of the false alarms on predictive accuracy by using two online classifiers, so Todi performed well even when false alarms occurred. Table 4.2

also shows that the error rate of Todi was the best.

4.3.5 Short Summary

Our new proposed method, Todi, uses two online classifiers: one that is reinitialized and the other one that is not reinitialized when concept drift is detected. Its predictive accuracy does not seriously deteriorate if a false alarm occurs, because Todi has a classifier that keeps a high accuracy immediately after the reinitialization of the other classifier. Using two classifiers can also remove the short-term memory required for storing training examples that conventional systems have. Furthermore, Todi is able to notify a user of the occurrence of concept drift accurately. Experimental results show that Todi performed well in learning and detecting concept drift in both synthetic and actual datasets.

4.4 Summary

To detect concept drift in a small number of examples, several methods have been recently proposed that monitor prediction errors of a single online classifier. These methods assume that a significant increase in the error rate for the online classifier suggests that the target concept is changing. Also, these methods do not depend on the attributes of the type of input in contrast to methods that estimate the underlying distribution of examples.

We proposed a method of more accurately detecting concept drift, which we call the STEPD method. STEPD monitors the predictive accuracy of a single online classifier and detects significant decreases in predictive accuracy, which are caused by concept drift, by using a statistical test of equal proportions. The online classifier is reinitialized to prepare for the learning of the next concept when concept drift is detected. Experimental results showed the statistical test enabled STEPD to detect various types

of concept drift in synthetic datasets more quickly and accurately than conventional systems. However, its false alarms worsen the predictive accuracy significantly because STEPD only uses a single online classifier.

To reduce the bad influence of false alarms on predictive accuracy, we developed a new system, the Todi system. Todi uses two online classifiers: one that is reinitialized and the other one that is not reinitialized when concept drift is detected. Todi uses the slightly modified STEPD method with one of the two online classifiers used to detect concept drift. Using two online classifiers also enables Todi to confirm the correctness of the last detection and to accurately notify a user of the occurrence of concept drift. To confirm the correctness, Todi compares the predictive accuracies of the two classifiers by using the same statistical test of equal proportions that STEPD uses. If the predictive accuracy of the reinitialized classifier was significantly better than the accuracy of the other classifier, Todi assumes that the last detection was correct. Experimental results showed Todi performed well in learning and detecting concept drift in a synthetic dataset. We then tackled a spam filtering problem, and the results demonstrated that Todi, with two Bogofilters, performed better than the single Bogofilter (Bogofilter is a well-known and successful spam filter).

One of our goals is to solve real-world problems for which the notification of the occurrence of changes is relevant. However, Todi does not consider the occurrence of recurring concepts, so we will combine Todi with the ACE system to create a more intelligent online learning system that can be used for solving various real-world problems.

Chapter 5

Detecting Sudden Concept Drift with a Method Inspired by Human Behavior

Machine learning methods for detecting concept drift that treat any misclassification as the same misclassification are disputable from a human behavioral perspective. We therefore studied human change detection [69]. In Section 5.1, we first describe our working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. We then discuss the human behavior experiment that we conducted to investigate our working hypothesis in Section 5.2. Experimental results suggested the validity of our working hypothesis, and we noticed that the successive rejection of highly confident classifications is significant for change detection. Using this knowledge, we thought the LIF model, which is a spike generation model, could be applied to change detection. In Section 5.3, we explain our proposed *l*eaky *i*ntegrate-and-*d*etect (LID) model that is based on LIF. LID uses the degree of confidence in and the recent accuracy of an online classifier to detect sudden and significant changes. Computer experiments given in Section 5.4 show that LID was able to respond to sudden and significant changes quickly and accurately. Finally, we summarize this study in Section 5.5.

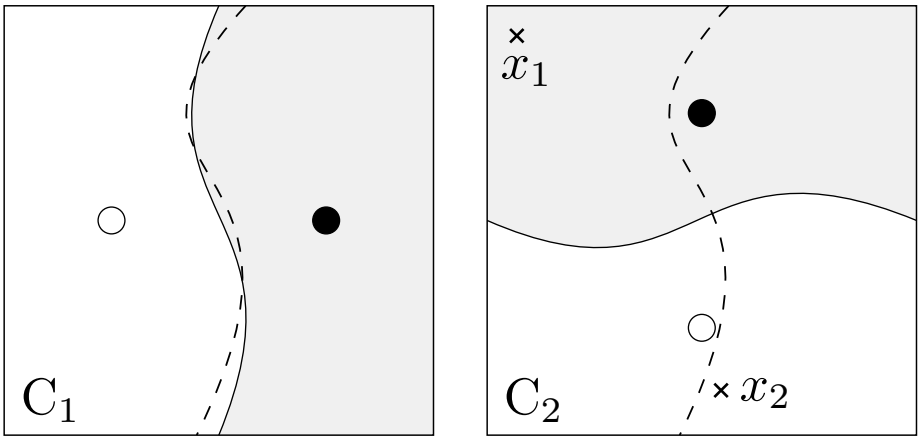

Fig. 5.1 An example of concept drift. The solid lines are the true boundary of C_1 and C_2 between classes ∘ and •, and the dotted line is the boundary learned from C_1. Both inputs x_1 and x_2 are misclassified immediately after the change from C_1 to C_2.

5.1 Working Hypothesis

Now we consider that the target concept is suddenly changed from a concept C_1 to a concept C_2, as shown in Figure 5.1. In the case where an input x_1, which is far from the true boundary of C_1, is given immediately after the change, humans classify x_1 into class ∘ confidently; however, this is wrong in C_2. In the case where an input x_2, which is close to the true boundary, is given instead of x_1, humans classify x_2 into class • without certainty; also, this is wrong in C_2. It is difficult to think that humans treat both cases equally regarding their change detection. However, as we have mentioned in Chapter 4, most machine learning methods for detecting concept drift treat the both cases as the same misclassification. There is significant difference between human methods and such machine learning methods.

We thought that humans may detect changes according to the occurrence of a rare event. In classification problems, the rare event would mean the rejection of confident classifications after they have completed learning the target concept. We therefore conducted a human behavior experiment to investigate the working hypothesis that

humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high.

5.2 Human Behavior Experiment and Results

In this section, we first describe an experimental method. We then present experimental results, and then discuss the results. Finally, we briefly summarize this human behavior experiment.

5.2.1 Experimental Method

We first describe the target concepts that we used in this experiment. Next, we present the experimental system that we used. We then explain our experimental procedures and describe the experimental conditions that divided our subjects into three groups. And finally, we provide information about our subjects.

A Target Concepts

We focused on the human method for detecting sudden changes and used the Stagger Concepts that are a well-known pattern classification problem to evaluate machine learning systems for handling sudden changes [87, 93]. They consist of 27 kinds of figures that have three attributes: *color* $\in$ {red, green, blue}, *shape* $\in$ {circle, square, triangle}, and *size* $\in$ {small, medium, large}. We prepared the following three concepts (see Figure 5.2) that are different from the original concepts,

(a) C_1 [*color* = blue and *size* = large] or [*shape* = circle and *size* = small] or [*color* = green and *shape* = triangle] or [*color* = red and *shape* = triangle],

(b) C_2 [*shape* = square] or [*color* = red and *shape* = circle],

(c) C_p [*color* = green and *size* = small] or [*shape* = triangle and *size* = large],

(a)	Circle			Square			Triangle		
	S	M	L	S	M	L	S	M	L
Red	■						■	■	■
Green	■						■	■	■
Blue	■		■			■			■

(b)	Circle			Square			Triangle		
	S	M	L	S	M	L	S	M	L
Red	■	*	*	*	*	*	*	*	*
Green	*			*	*	*	*	*	*
Blue	*		*	*	*	■			*

(c)	Circle			Square			Triangle		
	S	M	L	S	M	L	S	M	L
Red									■
Green	■			■			■		■
Blue									■

Fig. 5.2 Target concepts. (a) C_1, (b) C_2, and (c) C_p (for practice). The target concept changes from C_1 to C_2. 'S', 'M', and 'L' indicate "small", "medium", and "large", respectively. Gray cells indicate figures belonging to class A. Cells with '$*$' indicate figures of which the class in C_1 is different from the class in C_2.

where C_p is a concept for practice. The target concept changes from C_1 to C_2. Figures belong to class A if the figures follow each rule and class B otherwise. Note that we used this difficult C_1 to prevent subjects from completing the learning of C_1 too quickly.

B Experimental System

We designed the human behavior experiment with a software program that we made. Figure 5.3 shows the experimental screen of the program. The subjects are instructed to "... correctly classify figures into class A or class B". The figures are displayed in the lower left of the screen (Figure 5.3 (i)). The detailed information of the currently presented figure is displayed in the upper right of the screen (Figure 5.3 (ii)). The subjects classify the figure by clicking on an classification button (Figure 5.3 (iii)) using a generic mouse. Note that the subjects click on the lower buttons labeled "A!!" or

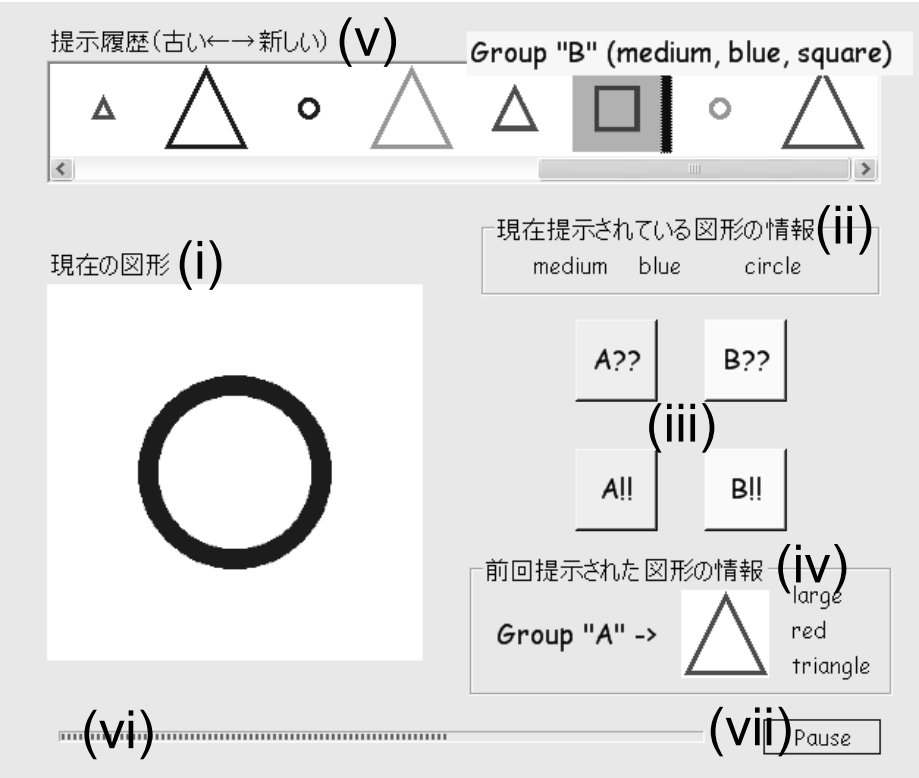

Fig. 5.3 The experimental screen. Numbers (i) thorough (vii) indicate the currently presented figure, its information, classification buttons, the last presented figure, the presented figures history (the last 20 figures), the progress bar of classification time (10 sec), and the pause button, respectively.

"B!!" when they have confidence in the classification. In contrast, the subjects click on the upper buttons labeled "A??" or "B??" when they do not have confidence in the classification[†5.1]. The correct class is displayed on the entire screen for 300 milliseconds after the classification. After the presentation of the correct class, three events occur at the same time: the next figure is displayed; the mouse cursor is automatically moved toward the center of the four classification buttons; and all buttons are disabled for a short time interval. The time interval is randomly decided in the range of 300 to 900 milliseconds.

Moreover, the last presented figure (Figure 5.3 (iv)) and the history of the last 20 figures (Figure 5.3 (v)) are displayed, but the history disappears 10 seconds after

[†5.1] Because a large individual difference was found in the introspection reports of the subject with these classification buttons, we did not analyze these reports.

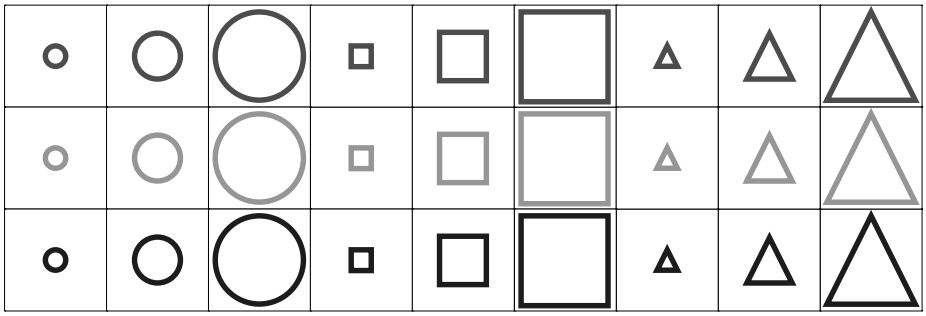

Fig. 5.4 Visually presented figures (3 colors (red, green, and blue) × 3 shapes (circle, square, and triangle) × 3 sizes (small, medium, and large) = 27 kinds).

the presentation of each figure (Figure 5.3 (vi)). If the currently presented figure is contained in the history, its information is removed from the history temporarily. The subjects are able to rest freely by clicking on the pause button (Figure 5.3 (vii)).

This experimental system records the reaction time, which is the time from the presentation of a figure to subject classification by clicking on an button, and records the history of classifications. Moreover, the system regards all 27 kinds of figures, which are shown in Figure 5.4, as a block. The system presents figures in random order for each block, except for the first block after the change of the target concept.

C Experimental Conditions

To investigate the working hypothesis, we used three experimental conditions and thus divided the subjects into three groups. The experimental system controls the time at which the target concept changes with the recent predictive accuracy of each subject, and controls the presentation order in the first block after the change with the degree of confidence that each subject has for figures.

Recent Accuracy The experimental system changes the target concept from C_1 into C_2 (Figure 5.2 (a), 5.2 (b)) when the accuracy for the most recent 27 figures exceeds a threshold θ. The value of θ is different between Condition 1 ($\theta = 0.90$) and Conditions 2 and 3 ($\theta = 0.75$). The number of presented figures in C_1 is different among the subjects. The concept C_2 has 135 figures (5 blocks).

Degree of Confidence The experimental system presents figures in the first block after the change in descending (Conditions 1 and 2) or ascending (Condition 3) order of the degree of confidence in the figures.

The degree of confidence is a psychological quantity that indicates the strength of self-confidence in one's own answer, so we cannot measure its true value. We thought that the degree of confidence in figures has the following properties: (1) the degree of confidence in a figure is dependent on the ease of the classification of the figure; (2) the degree of confidence in a figure is modified by the feedback on the correct class of the figure; (3) humans confidently classify a figure that they were able to correctly classify in past classification opportunities for the figure; and (4) new feedback for a figure has an effect on the modification of the degree of confidence in the figure more than old feedback for the figure.

According to these properties, we defined the *order* of the degree of confidence in all 27 figures. First, we divided the 27 figures into two groups: figures that were classified correctly in the last classification opportunity and the other figures. Next, for each group, we decided the order of the degree of confidence using the accuracy for each figure from the beginning of the experiment. And finally, we defined the degree of confidence in all figures of the former group to be higher than the degree of confidence in any figure of the latter group. We think that the degree of confidence that humans actually have is the result of more complicated processes. However, this rough order was sufficient to conduct this experiment.

To summarize, we used three experimental conditions as shown in Figure 5.5[†5.2]

- Condition 1 The target concept changes from C_1 into C_2 when the accuracy for the most recent 27 figures exceeds the threshold $\theta = 0.90$ ($> 24/27$), and figures in the first block after the change are presented in the *descending* order

[†5.2] From the results of preliminary experiments, the subjects confidently classified almost all of the 27 figures when the recent accuracy was higher than 0.90. Therefore, we eliminated the condition that "$\theta = 0.90$ and the ascending order."

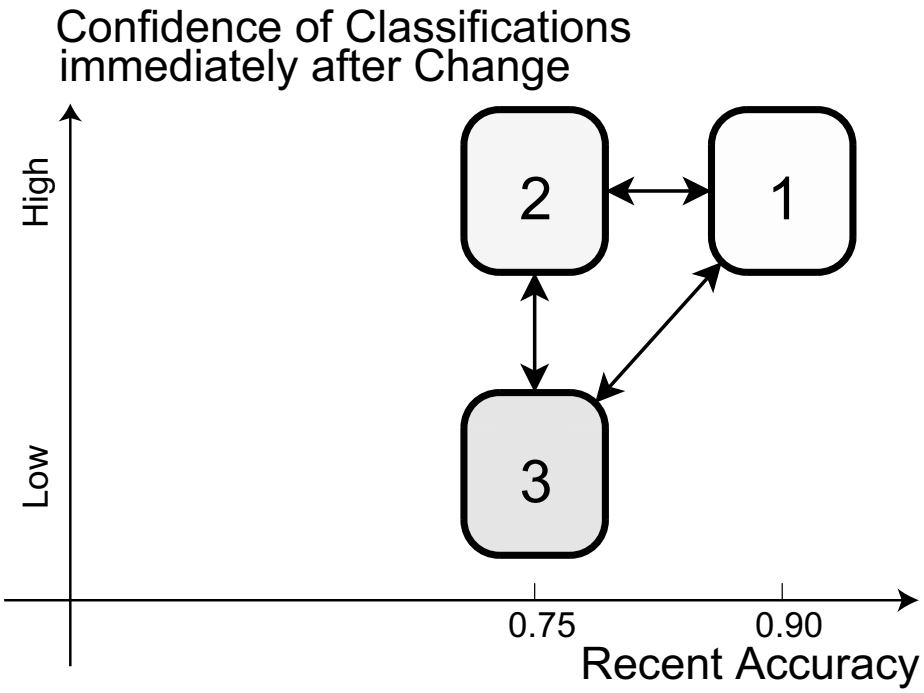

Fig. 5.5 Three experiment conditions. We investigated the working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high.

of the degree of confidence in the figures.

- Condition 2 $\theta = 0.75$ ($> 20/27$) and the *descending* order.
- Condition 3 $\theta = 0.75$ ($> 20/27$) and the *ascending* order.

By using these conditions, we investigated the working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. Let c_1, c_2, and c_3 be Conditions 1, 2, and 3.

D Experimental Procedure

The subjects first had practice using the concept shown in Figure 5.2 (c). Each subject finished practicing when each subject chose the correct class of figures 27 times successively. The subjects then started the experiment with one experimental condition. We did not inform the subjects that the target concept was going to change.

E Subjects

The subjects were 41 undergraduate and graduate students (31 men and 10 women from 18 to 30 [ave. 21.5] years of age). We first eliminated the results of three subjects. The accuracy for these three subjects for the most recent 27 figures in C_1 did not exceed the threshold θ until 405 figures were presented in C_1. We then eliminated the results of another three subjects. The accuracy for these three subjects for the most recent 27 figures in C_2 did not exceed 15/27 until 81 figures were presented in C_2[†5.3].

The number of the subjects that we analyzed their results is the following 35 people:

- Condition 1 10 people (8 men and 2 women, 19 – 25 [ave. 21.5] years of age).
- Condition 2 13 people (11 men and 2 women, 19 – 23 [ave. 21.5] years of age).
- Condition 3 12 people (9 men and 3 women, 18 – 25 [ave. 21.7] years of age).

5.2.2 Experimental Results

To carefully analyze the behavior of the subjects immediately after the change of the target concept, we divided one block (27 figures) into three sub-blocks (9 figures × 3). Moreover, we eliminated figures from the sub-blocks if the figures did not satisfy all following conditions: (1) the class of a figure in C_1 was the same as the class of the figure in C_2; (2) the subjects clicked on the "pause" button during the presentation of a figure; and (3) the reaction time, t, for a figure was greater than 14.0 seconds [†5.4].

The z-score of the reaction time of each subject is calculated by $Z = (t - \mu_{40})/\sigma_{40}$, where μ_{40} and σ_{40} are the mean and the standard deviation of the reaction time of each subject for the first 40 figures.

[†5.3] 15/27 was nearly equal to $\mu - 2.16\sigma$ and 16/27 was nearly equal to $\mu - 1.90\sigma$, where μ and σ are the mean and the standard deviation of the maximum accuracy for the most recent 27 figures in C_2 until 81 figures were presented in C_2 over the 38 subjects. $\mu = 0.861$ and $\sigma = 0.141$.

[†5.4] 14.0 was nearly equal to $\mu' + 1.94\sigma'$, where μ' and σ' are the mean and the standard deviation of the reaction time for the first 40 figures over the 35 subjects. $\mu' = 5.66$ and $\sigma' = 4.30$.

Figures 5.6 (a) and 5.6 (b) show the accuracy and the z-score of reaction time before and after the change of the target concept. Sub-blocks -3 through -1 indicate sub-blocks before the change, and sub-blocks 1 through 6 indicate sub-blocks after the change. Let b_i be the ith sub-block. Note that the next sub-block of b_{-1} is b_1.

A Accuracy

First, we analyzed the results of the accuracy with a two-factor (condition (c_1, c_2, c_3) $\times$ sub-block $(b_{-3}, \ldots, b_{-1},\ b_1, \ldots, b_6)$) analysis of variance (ANOVA [55, 67]) with repeated measures on one factor (sub-block). The interaction between the condition and the sub-block was significant ($F[16, 256] = 3.896,\ p < .001$). The main effect of the condition was not significant ($F[2, 32] = 1.657$, n.s.) and the main effect of the sub-block was significant ($F[8, 256] = 55.81,\ p < .001$).

Next, we analyzed the results with a one-way ANOVA with repeated measures for each condition. The simple effect of the sub-block at each condition was significant ($F[8, 72] = 35.20,\ p < .001$ at c_1; $F[8, 96] = 21.28,\ p < .001$ at c_2; and $F[8, 88] = 12.94$, $p < .001$ at c_3). We then used the Bonferroni Test as a post-hoc test [55]. The accuracy significantly increased from b_1 to b_2 at c_1 ($p < .001$), from b_1 to b_3 at c_2 ($p < .01$), and from b_1 to b_4 at c_3 ($p < .001$). Also, the accuracy significantly decreased from b_{-1} to b_1 at each condition ($p < .001$).

Finally, we analyzed the results with a one-way ANOVA for each sub-block after the change. The simple effect of the condition at b_1 was significant ($F[2, 34] = 9.596$, $p < .001$). We then used the Bonferroni Test as a post-hoc test. The accuracy significantly differed between c_1 and c_3 at b_1 ($p < .001$) and between c_2 and c_3 at b_1 ($p < .05$).

B Z-score of reaction time

First, we analyzed the results of the z-score of reaction time with a two-factor ANOVA with repeated measures on one factor. The interaction between the condition

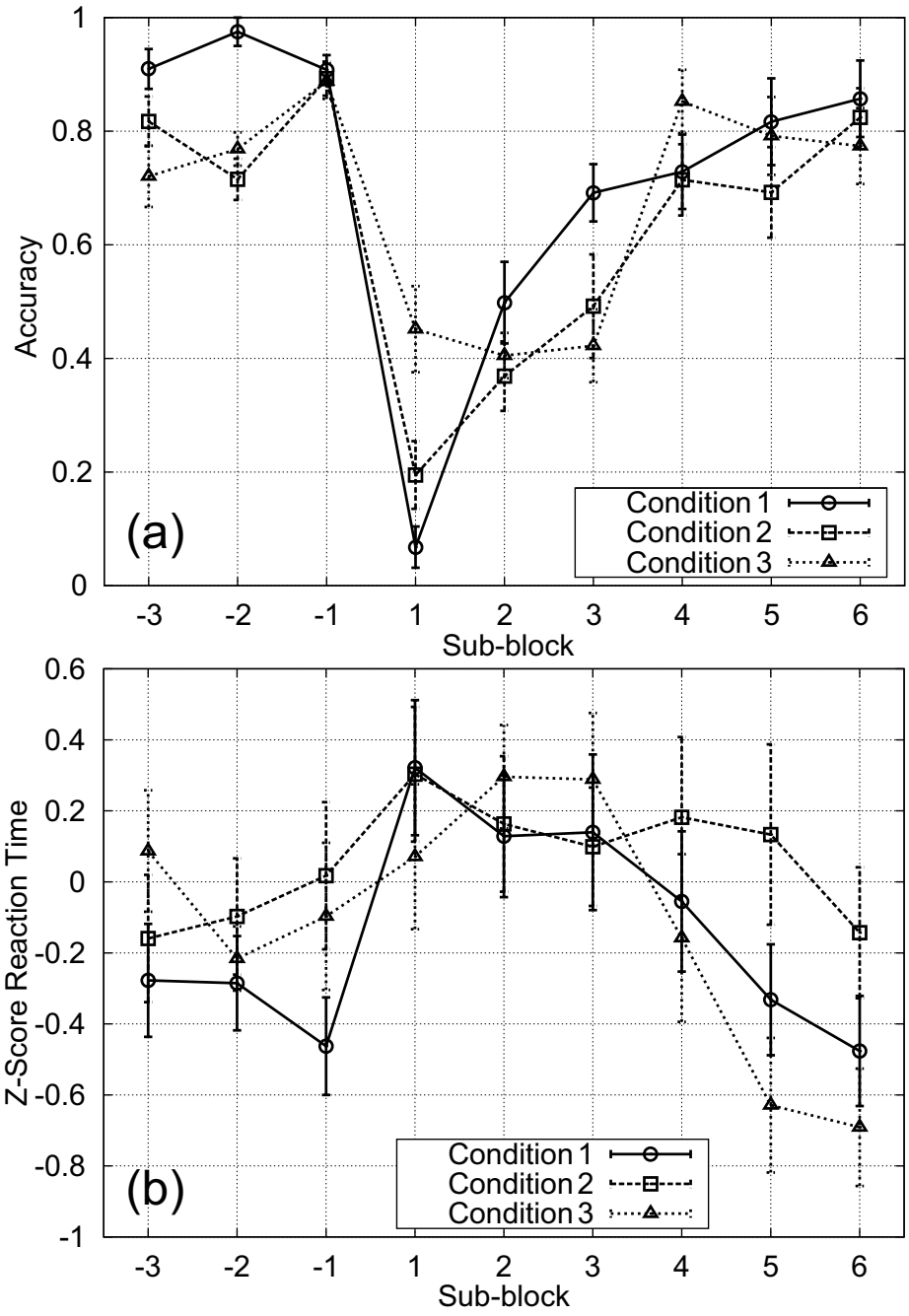

Fig. 5.6 (a) The accuracy and (b) the z-score of reaction time before and after the change of the target concept. Error bars indicate standard errors. Sub-blocks -3 through -1 indicate sub-blocks before the change, and sub-blocks 1 through 6 indicate sub-blocks after the change.

and the sub-block was significant ($F[16, 256] = 2.136$, $p < .01$). The main effect of the condition was not significant ($F[2, 32] = 0.6271$, n.s.) and the main effect of the sub-block was significant ($F[8, 256] = 8.749$, $p < .001$).

Next, we analyzed the results with a one-way ANOVA with repeated measures for each condition. The simple effect of the sub-block at c_1 and at c_3 was significant ($F[8,72]=6.161$, $p<.001$ at c_1; $F[8,96]=1.618$, n.s. at c_2; and $F[8,88]=5.777$, $p<.001$ at c_3). We then used the Bonferroni Test as a post-hoc test. The z-score of reaction time significantly decreased from b_{-1} to b_1 at c_1 ($p<.001$). Also, the z-score significantly decreased from b_{-1} to b_5 at c_1 ($p<.01$) and from b_2 to b_5 at c_3 ($p<.01$).

Finally, we analyzed the results with a one-way ANOVA for each sub-block after the change. The simple effect of the condition at b_5 was significant ($F[2,34]=3.478, p<.05$ at b_5; $F[2,34]=1.563$, n.s. at b_1). We then used the Bonferroni Test as a post-hoc test. The z-score significantly differed between c_2 and c_3 at b_5 ($p<.05$).

5.2.3 Discussion

Note again that the target concept changes from C_1 into C_2 when the accuracy for the most recent 27 figures exceeds the threshold $\theta = 0.90$ (at the condition c_1) or 0.75 (at c_2 and at c_3). Moreover, figures are presented to the subjects in the first block after the change (in the sub-blocks b_1, b_2, and b_3), in descending (at c_1 and at c_2) or ascending (at c_3) order of the degree of confidence in the figures.

Although the accuracy significantly increased from b_1 to b_2 at c_1 and from b_1 to b_3 at c_2, the accuracy did not increase during the three sub-blocks b_1, b_2, and b_3 at c_3. The subjects at c_3 did not have an opportunity to reject their confident classifications immediately after the change, and thus they were not able to detect the change quickly. Moreover, the accuracy significantly differed between c_1 and c_3 at b_1 and between c_2 and c_3 at b_1. Although the main effect of the condition was not significant, the transitions of the accuracy at c_1 and at c_2 were different from the transition at c_3. We therefore consider that the degree of confidence has an effect on human change detection.

The z-score of reaction time increased significantly only from b_{-1} to b_1 at c_1. We

think that the successive rejection of highly confident classifications caused the significant increase at c_1. In contrast, the z-score of reaction time did not significantly increase from b_{-1} to b_1 at c_2 and at c_3. Although the main effect of the condition was also not significant, the transition of the z-score of the reaction time at c_1 was different from the transitions at c_2 and at c_3. We therefore consider that the recent accuracy also has an effect on human change detection.

5.2.4 Short Summary

We conducted a human behavior experiment to confirm the working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. We designed this experiment to focus on the human method for detecting sudden changes. Experimental results suggested the validity of the working hypothesis.

Moreover, we noticed that the successive rejection of highly confident classifications is significant for change detection. Based on this finding, we have proposed a machine learning method for detecting sudden and significant changes, the LID model, as a stepping-stone to deal with various types of concept drift.

5.3 The LID model

Based on the findings that we got from the human behavior experiment, we thought that the *l*eaky *i*ntegrate-and-*f*ire (LIF) model, which is a spike generation model [48, 56], could be applied to change detection. In this section, we first present the LIF model and then describe our proposed *l*eaky *i*ntegrate-and-*d*etect (LID) model that is based on LIF. The aim of the LID model is to detect sudden and significant changes quickly and accurately in an environment that also includes noise and gradual changes.

5.3.1 LIF (Leaky Integrate-and-Fire) Model

Figure 5.7 shows the LIF model [48, 56]. In the LIF model, the evolution of the membrane potential, $V(t)$, is described by the following first-order differential equation,

$$\begin{cases} \tau \dfrac{dV(t)}{dt} + V(t) - I(t) = 0 \\ V(t) = V_{\text{rest}} \quad \text{if } V(t) > V_{\text{th}} \end{cases}, \tag{5.1}$$

where $I(t)$, V_{th}, and V_{rest} indicate the injected current at time t, a threshold and a resting membrane, respectively. Upon the arrival of the injected current $I(t)$ at time t, $V(t)$ is instantaneously updated by $V(t) + \Delta V$. $V(t)$ decays exponentially with a time constant τ toward V_{rest} until a new injected current arrives. If $V(t)$ crosses the threshold V_{th}, a spike is formed and $V(t)$ is reset to a resting value V_{rest}.

5.3.2 Applying LIF to Detecting Sudden Concept Drift

The findings that we got from the human behavior experiment is that the successive rejection of highly confident classifications is significant for change detection. Using this finding, we thought that the LIF model could be applied to change detection. We regarded the membrane potential that was increased by injected currents as a state value increased by misclassifications. The integration of the state value with time decay reduces false alarms, which are caused by noise and gradual changes, and enables detecting frequent occurrences of misclassifications, which are caused by sudden and significant changes. We thus proposed the *l*eaky *i*ntegrate-and-*d*etect (LID) model that is based on the LIF model. LID uses a single online classifier.

We first rewrote Equation (5.1) as the recurrence equation,

$$V_{t+1} = \gamma V_t + I_t, \tag{5.2}$$

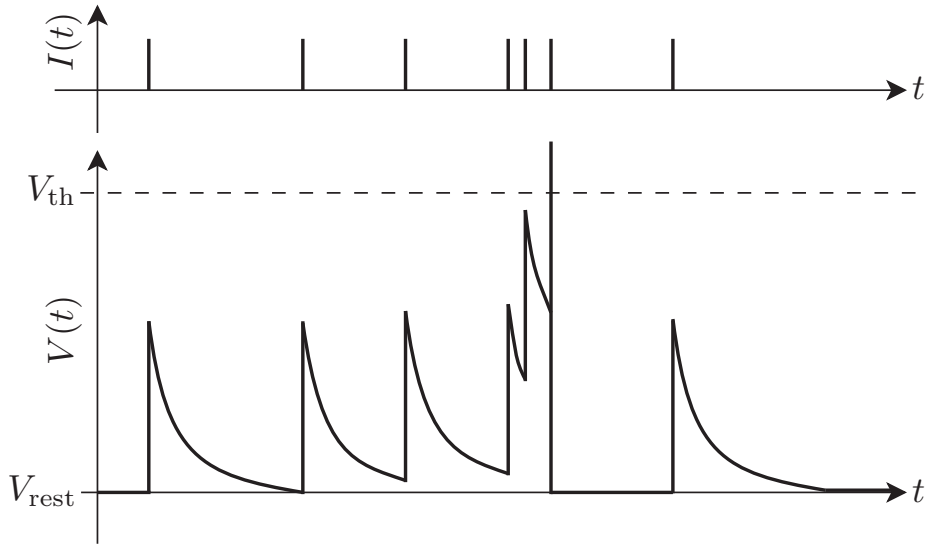

Fig. 5.7 The LIF model. $I(t)$, $V(t)$, V_{th}, and V_{rest} indicate the injected current at time t, the membrane potential at time t, a threshold, and a resting membrane potential, respectively. $V(t)$ decays exponentially with a time constant τ toward V_{rest} until a new injected current arrives.

where V_t, I_t, and γ are the state value at time t, the input value at time t, and a decay constant, respectively. The input value at time t is

$$I_t = \begin{cases} g(A_t, C_t) & \text{if } H(\boldsymbol{x}_t) \neq y_t \\ 0 & \text{otherwise} \end{cases}, \tag{5.3}$$

where $A_t \in (0, 1)$ is the recent accuracy for the most recent W examples, $C_t \in [0, 1]$ is the degree of confidence in the current input $\boldsymbol{x}_t$, and the input function is

$$g(A_t, C_t) = A_t C_t. \tag{5.4}$$

Thus, I_t reflects our working hypothesis. That is to say, I_t has a large value when the classifier output $H(\boldsymbol{x}_t)$ that has the high degree of confidence C_t is rejected despite the fact that the recent accuracy A_t is high. If V_{t+1} exceeds a threshold, V_{th}, the LID model assumes that a sudden change occurred. At this time, the online classifier is reinitialized, and V_{t+1} is set to 0 ($V_{\text{rest}} = 0$).

The recent accuracy is given by Laplace's rule of succession [57],

$$A_t = \frac{s+1}{W+2}, \tag{5.5}$$

where s is the number of correct classifications of the online classifier for the most recent W examples. As we mentioned before, this definition achieves a better estimation of the successful classification rate than simply dividing s by W (maximum likelihood estimate), especially when sample sizes (W) are small [59].

We defined the degree of confidence as

$$C_t = 1 - \frac{-\sum_i c_i \log_2 c_i}{\log_2 Y}, \tag{5.6}$$

where Y is the number of classes, and c_i is the output value for the ith class by the online classifier for the current input $\boldsymbol{x}_t$ (satisfying $\sum_{i=1}^{Y} c_i = 1$ and $H(\boldsymbol{x}_t) = \arg\max_i c_i$).

LID can use any online classifier only if the classifier can calculate the output values satisfying $\sum_{i=1}^{Y} c_i = 1$. Moreover, LID does not need to store training examples. The pseudo-code of LID is shown in Algorithm 5.1.

5.4 Computer Experiments and Results

We used two synthetic datasets in the computer experiments to evaluate the change detection of LID. The datasets include sudden and significant changes. In this section, we first present the experimental results and then discuss future work.

5.4.1 SINE2

We first used the SINE2 problem [31, 51] to provide empirical support for the effectiveness of LID for sudden and significant changes. In this problem, examples

Algorithm 5.1 LID (leaky integrate-and-detect) model.

Parameters: W: window size, V_{th}: threshold, γ: decay const.
Online Classifier: $H : X \to Y$.
Initialize state value $V_0 \leftarrow 0$, $s \leftarrow 0$, $\{w\} \leftarrow \Phi$.
for each example ($\boldsymbol{x}_t \in X$, $y_t \in Y$) **do**
  Output: $H(\boldsymbol{x}_t)$ with output value for each class $\{c_i \mid \sum_{i=1}^{Y} c_i = 1\}$.
  set $w_t \leftarrow [\![H(\boldsymbol{x}_t) = y_t]\!]$. // **true** or **false**.
  if w_t is **true then** // output is correct
    increment s.
  end if
  if w_{t-W} is **true then** // window overflows.
    decrement s.
  end if
  train online classifier H with $(\boldsymbol{x}_t, y_t)$.
  set $A_t \leftarrow (s+1)/(W+2)$. // recent accuracy
  set $C_t \leftarrow 1 - (-\sum_i c_i \log_2 c_i)/(\log_2 Y)$. // degree of confidence
  if w_t is **false then** // output is wrong
    update $V_{t+1} \leftarrow \gamma V_t + g(A_t, C_t)$, where $g(A_t, C_t) = A_t C_t$.
  end if
  if $V_{t+1} > V_{\mathrm{th}}$ **then** // sudden change occured.
    reinitialize online classifier H; reinitialize V_{t+1}, $s \leftarrow 0$, $\{w\} \leftarrow \Phi$
  end if
end for

that have two continuous inputs $(x_1, x_2) \in [0, 1]^2$ are labeled satisfying

$$x_2 < 0.5 + 0.3 \sin(3\pi x_1) \tag{5.7}$$

as positive. A sudden and significant change was introduced by reversing the sign of inequality in Equation (5.7) at time 500. At each time step, an online learning system was trained with one example and then was tested with 100 examples generated randomly according to the current concept. The total number of training examples was 1000. There was no noise and there were no gradual changes.

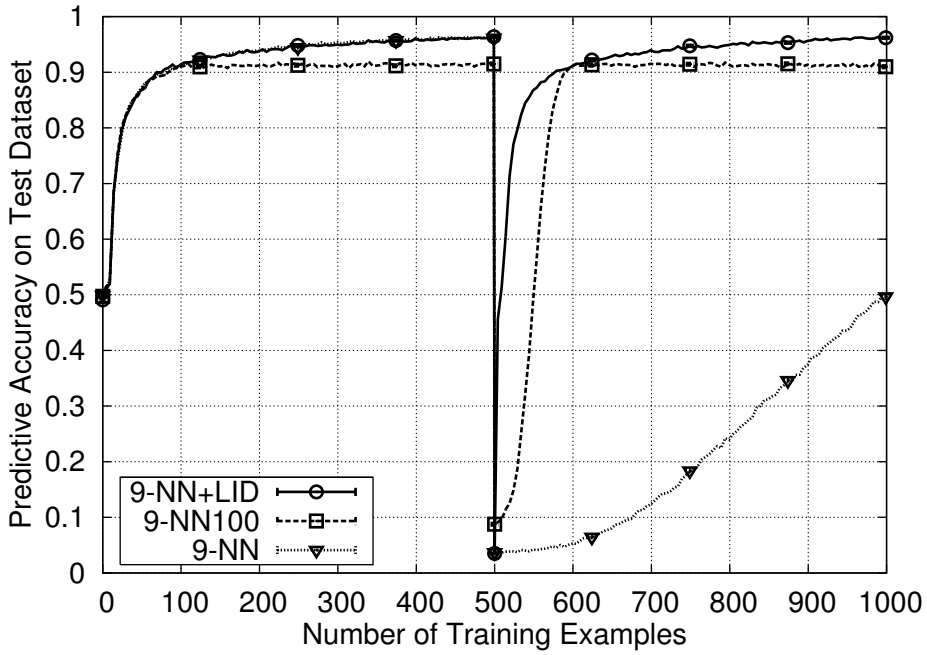

Fig. 5.8 Predictive accuracies of 9-Nearest Neighbors (9-NN), LID with 9-NN (9-NN+LID), and the 9-NN that adapts to the most recent 100 examples (9-NN100) in the SINE2 problem. Results were averaged over 200 trials. Error bars indicate standard errors.

A Predictive Accuracy of LID with k-Nearest Neighbors

We used a k-Nearest Neighbors (k-NN) for the online classifier of LID in this experiment. The output value of k-NN for each class in Equation (5.6) is

$$c_i = \frac{\sum_{(\boldsymbol{x}_j, y_j) \in \boldsymbol{X}_t^k} [\![y_j = i]\!]}{k} \quad (i = 0, 1), \tag{5.8}$$

where $\boldsymbol{X}_t^k$ is k nearest neighbors for the input $\boldsymbol{x}_t$ at time n, and $[\![y_j = i]\!]$ is 1 if $y_j = i$ and 0 otherwise.

First, LID with 9-NN (9-NN+LID) was compared with the original 9-NN and the 9-NN that adapts to the most recent 100 examples (9-NN100). The parameter settings for LID were $W = 100$, $V_{\mathrm{th}} = 2.5$, and $\gamma = 0.99$. Figure 5.8 shows the results of their predictive accuracies.

9-NN performed poorly because it does not consider the occurrence of changes. 9-

Table. 5.1 False alarm rates and numbers of required examples for detecting the first change for LIDs and DDM on the first 600 examples in the SINE2 problem. Results were averaged over 1000 trials.

Detection Method	Parameter	False alarm Rate	No. of Required Examples
LID	$W = 50$	0.001	3.59 ± 0.0298
	$W = 100$	0.001	3.48 ± 0.0294
LID-A	$W = 50$	1.000	—
	$W = 100$	1.000	—
LID-C	—	0.124	3.28 ± 0.0289
LID-Const	$\kappa = 0.15$	0.010	14.8 ± 0.0560
	$\kappa = 0.16$	0.026	13.5 ± 0.0567
DDM	—	0.015	13.6 ± 0.0467

Notes: All methods used 9-NN. The $g(A_n, C_n)$ of LID, LID-A, LID-C, and LID-Const are A_nC_n, A_n, C_n, and κ. A_n is the accuracy for recent W examples at time n, C_n is the confidence for the input $\boldsymbol{x}_n$, and κ is a constant. Each element in the rightmost column indicates "mean $\pm$ standard error" from 1000 trials, except for trials where false alarms occur. Misdetection rates of all methods were 0.

NN100, which uses a window size of 100, was able to respond to the change, but the small window worsened the predictive accuracy of 9-NN100. A bigger window improves the predictive accuracy, but it delays the response to the change. Figure 5.8 shows that 9-NN+LID achieved both the high predictive accuracy and the quick response to the change. The reinitialization of the online classifier is effective in responding to sudden and significant changes.

B Change Detection of LID with k-Nearest Neighbor

We used these following four input functions to evaluate the input function of LID, $g(A_t, C_t) = A_tC_t$. This input function uses both the degree of confidence in and the recent accuracy of the online classifier according to the working hypothesis.

1. $g(A_t, C_t) = A_tC_t$ (LID)
2. $g(A_t, C_t) = A_t$ (LID-A)
3. $g(A_t, C_t) = C_t$ (LID-C)
4. $g(A_t, C_t) = \kappa$ (LID-Const; This κ is a constant.)

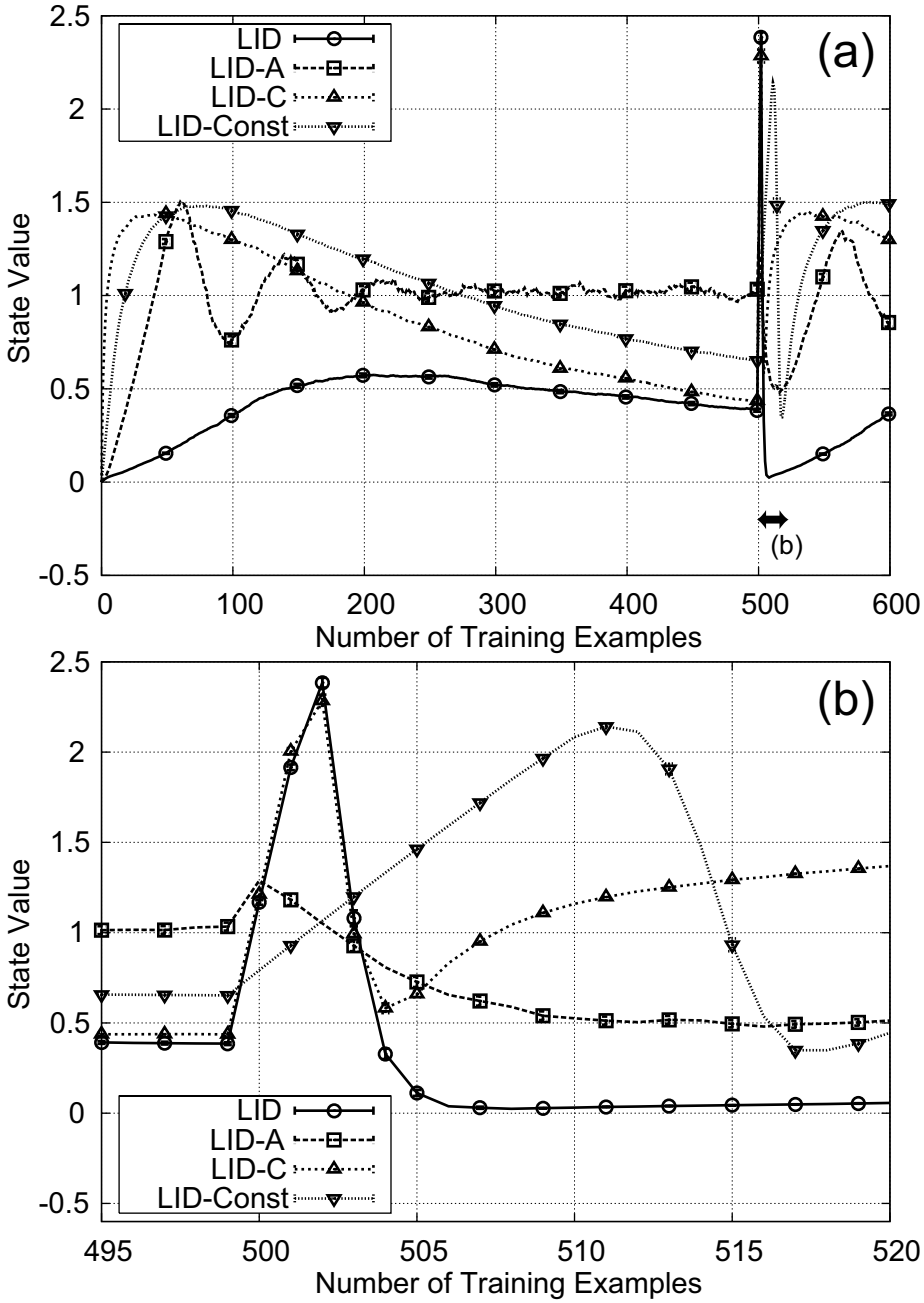

Fig. 5.9 (a) State values, V_t, of LIDs on the first 600 examples in the SINE2 problem. (b) The partially enlarged view ($W = 100$ and $\kappa = 0.15$). Results were averaged over 1000 trials. Error bars indicate standard errors.

In addition, we used a conventional method, DDM [31].

By using the first 600 examples from the SINE2 problem, we investigated their false alarm rates and their numbers of required examples for detecting the first change over 1000 trials. The parameter settings for LIDs were $V_{\rm th} = 2.5$ and $\gamma = 0.99$. Table 5.1

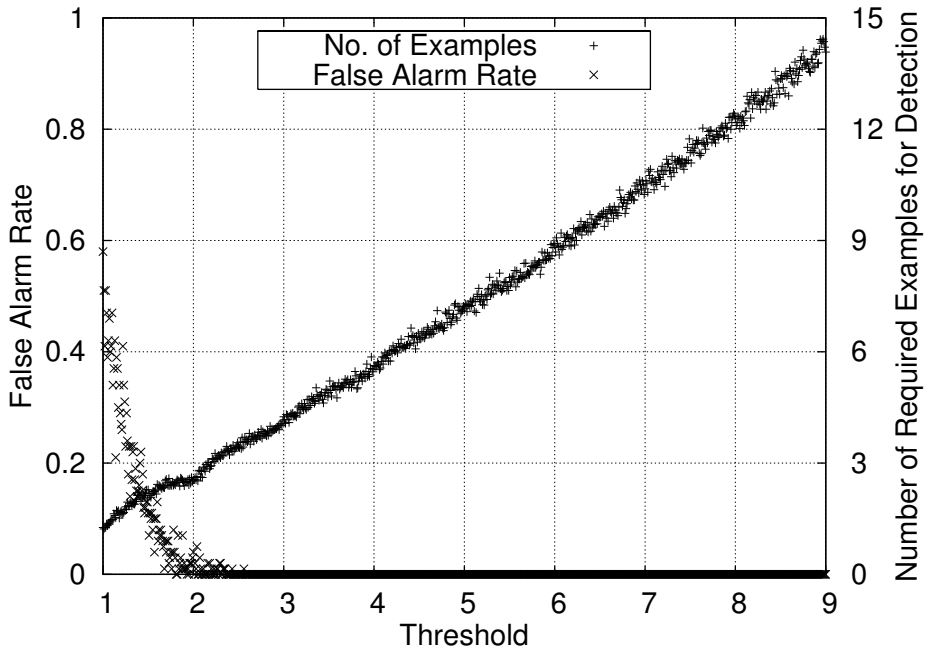

Fig. 5.10 False alarm rates and numbers of required examples for detecting the first change for LID on the first 600 examples in the SINE2 problem. V_{th} (threshold) $\in [1, 9]$, $W = 100$, and $\gamma = 0.99$. Results were averaged over 100 trials. Misdetection rates for all of the thresholds were 0.

show the results. Misdetection rates for all of the methods were 0. Note that "false alarm" means that the change is detected before time 500 and "misdetection" means that the change is not detected until time 600 in this experiment.

LID-A, which does not use the degree of confidence, C_t, repeated false alarms because this problem is apt to misclassify inputs close to the true boundary. LID-C, which does not use the recent accuracy, A_t, achieved the fastest detection, but LID-C caused false alarms frequently in the early stages of learning. Moreover, LID-Const needs a large value of κ to detect the change quickly, but such a value caused many false alarms. In contrast, LID, which uses both C_t and A_t, achieved quick and accurate detection. Using A_t enabled LID to avoid the significant increase of the state value in the early stages of learning. And then, using C_t enabled LID to detect sudden changes quite quickly with few false alarms (see Figure 5.9). Moreover, LID performed better than DDM because it was able to detect the change more quickly and accurately than DDM, which is a conventional method.

We then investigated LID's false alarm rate and its number of required examples for detecting the change while changing the threshold, V_{th}, from 1.00 to 9.00 at intervals of 0.01 ($W = 100$ and $\gamma = 0.99$). Figure 5.10 shows that the false alarm rate was less than 3% when V_{th} was greater than 2.07, and no false alarm occurred when V_{th} was greater than 2.56. Note that no misdetection occurred for any of the thresholds. In this problem, we were able to find the appropriate parameters for LID easily.

5.4.2 Moving Hyperplane

We then created a concept drift situation by using the Moving Hyperplane problem, which is a standard benchmark for concept drift [41, 92, 96], to provide empirical support for the performance of LID in an environment that also includes noise and gradual changes.

In this problem, examples uniformly distributed in multidimensional space $\boldsymbol{x} \in [0, 1]^d$ satisfying

$$\sum_{i=1}^{d} a_i x_i \geq a_0 \tag{5.9}$$

are labeled as positive. The weights of the moving hyperplane, $\{a_i\}$, which are initialized to $[-1, 1]$ randomly, are updated as

$$a_i \leftarrow a_i + s_i \frac{t}{N} \quad (i = 1 \ldots d) \tag{5.10}$$

at each time, where $s_i \in \{-1, 1\}$ is the direction of change for each weight. The threshold a_0 is calculated as

$$a_0 = \frac{1}{2} \sum_{i=1}^{d} a_i \tag{5.11}$$

at each time so that roughly half of the examples are positive. $\{s_i\}$ is reset randomly

every N examples. Sudden and significant changes were introduced by reversing the sign of inequality in Equation (5.9) every N examples. At each time step, an online learning system was trained with one example and then was tested with 100 examples generated randomly according to the current concept. The total number of training examples was 3000. Then, noise was introduced by switching the labels of 5% of the training examples. The parameters for this problem were $d = 10$, $t = 0.3$, and $N = 1000$.

A Predictive Accuracy of LID with Naive Bayes

In this experiment, we used the Naive Bayes (NB) that can handle numeric attributes [44, Equations (1) – (3)] for the online classifier of LID. The output value of NB for each class in Equation (5.6) is

$$c_i = \frac{p(i)p(\boldsymbol{x}_t|i)}{\sum_j p(j)p(\boldsymbol{x}_t|j)} \qquad (i = 0, 1), \tag{5.12}$$

LID with NB (NB+LID) was compared with the original NB and the NBs that adapt to the most recent 100 or 400 examples (NB100, NB400). The parameter settings for LID were $W = 100$, $V_{\text{th}} = 4.0$, and $\gamma = 0.99$. Figure 5.11 shows the results of their predictive accuracies.

NB performed poorly because it does not consider the occurrence of changes. The window size of 100 enabled NB100 to respond to the sudden and significant changes quickly, but the small window worsened the predictive accuracy of NB100. In contrast, the window size of 400 enabled NB400 to learn the target concept accurately, but the large window delayed the response of NB400 to the changes. Figure 5.11 shows that NB+LID achieved both the high predictive accuracy and the quick response to the sudden and significant changes in an environment that also includes gradual changes and noise.

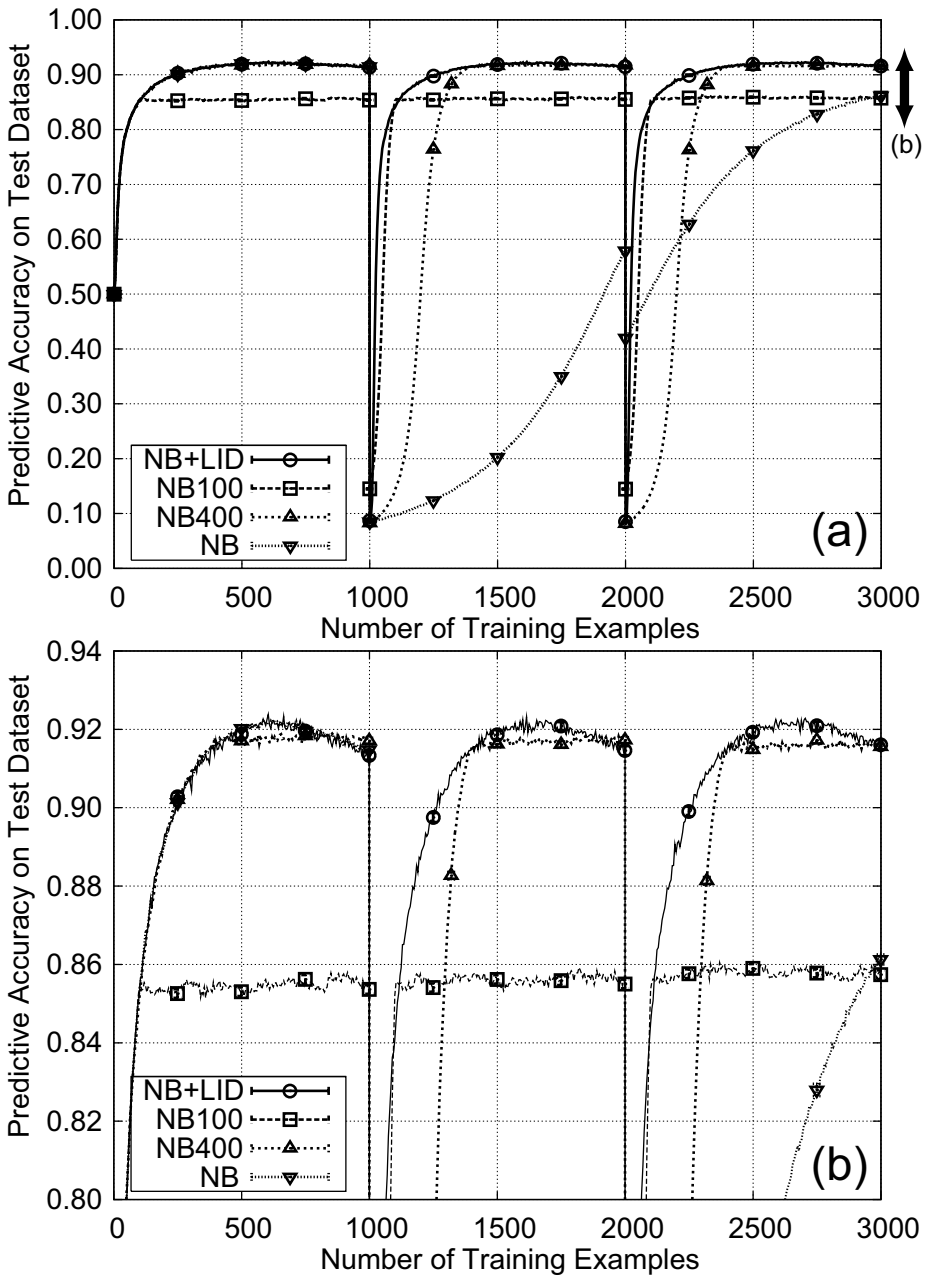

Fig. 5.11 (a) Predictive accuracies of Naive Bayes (NB), LID with NB (NB+LID), and the NBs that adapt to the most recent 100 or 400 examples (NB100, NB400) in the Moving Hyperplane problem. (b) The partially enlarged view. Results were averaged over 1000 trials. Error bars indicate standard errors.

B Change Detection of LID with Naive Bayes

We then compared LID with LID-A, LID-C, LID-Const, and DDM to evaluate the detection of LID in the Moving Hyperplane problem.

Table. 5.2 False alarm rates and numbers of required examples for detecting the first change for LIDs and DDM on the first 1200 examples in the Moving Hyperplane problem. Results were averaged over 1000 trials.

Detection Method	Parameter	False alarm Rate	No. of Required Examples
LID	$W = 50$	0.011	18.6 ± 0.250
	$W = 100$	0.007	16.4 ± 0.192
	$W = 200$	0.009	15.6 ± 0.181
LID-A	$W = 100$	1.000	—
LID-C	—	1.000	—
LID-Const	$\kappa = 0.16$	0.019	18.5 ± 0.127
	$\kappa = 0.17$	0.045	16.3 ± 0.126
DDM	—	0.022	29.2 ± 0.165

Notes: All methods were used with Naive Bayes. Each element in the rightmost column indicates "mean $\pm$ standard error" from 1000 trials, except for trials where false alarms occur. Misdetection rates of all methods were 0.

By using the first 1200 examples from the Moving Hyperplane problem, we investigated their false alarm rates and their numbers of required examples for detecting the first change over 1000 trials. The parameter settings for LIDs were $V_{\mathrm{th}} = 2.5$ and $\gamma = 0.99$. Table 5.2 show the results. Misdetection rates for all of the methods were 0. Note that "false alarm" means that the change is detected before time 1000 and "misdetection" means that the change is not detected until time 1200 in this experiment.

The Moving Hyperplane problem was more difficult than the SINE2 problem, so the state value of LID, V_t, increased quickly if the input value, I_t, was not given appropriately. LID-A and LID-C was therefore not able to detect the sudden and significant change at all, as shown in Figure 5.12. LID-Const performed well, but the false alarm rate of LID-Const was higher than that of LID. Moreover, the slight difference between values of κ considerably affected detection performance. Table 5.2 shows that LID was able to respond to the sudden and significant change quickly and accurately. The window size of 200 enabled LID to achieve the quickest detection with the low false alarm rate (0.9%).

We then investigated LID's error (false alarm and misdetection) rate and number of

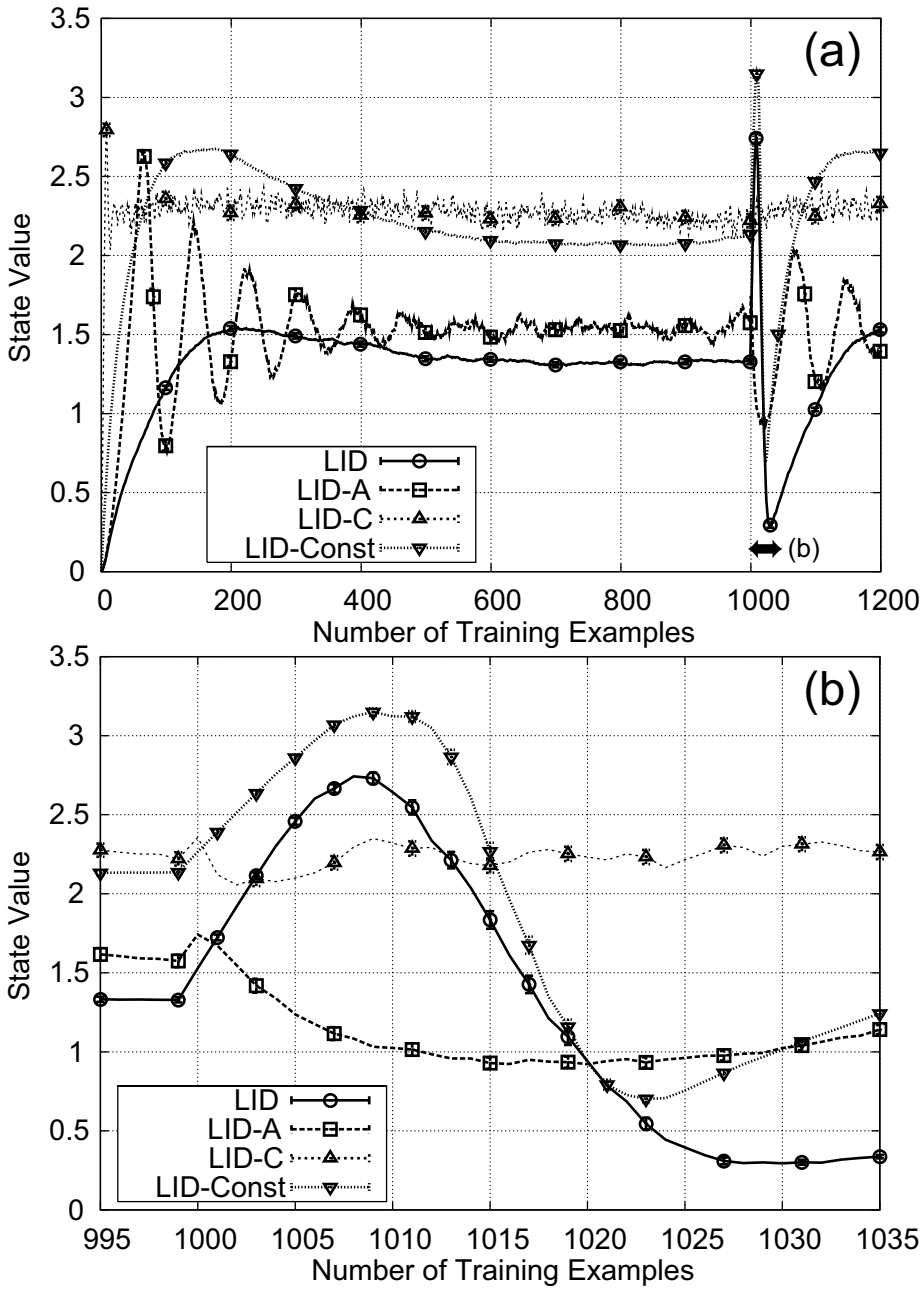

Fig. 5.12 (a) State values, V_t, of LIDs on the first 1200 examples in the Moving Hyperplane problem. (b) The partially enlarged view ($W = 100$ and $\kappa = 0.17$). Results were averaged over 1000 trials. Error bars indicate standard errors.

required examples for detecting the change while changing the threshold, V_{th}, from 1.00 to 9.00 at intervals of 0.01 ($W = 100$ and $\gamma = 0.99$). Figure 5.13 shows that the false alarms occurred when V_{th} was less than 4.0. In contrast, the misdetection occurred when V_{th} was greater than 6.0. In an environment that includes noise and gradual

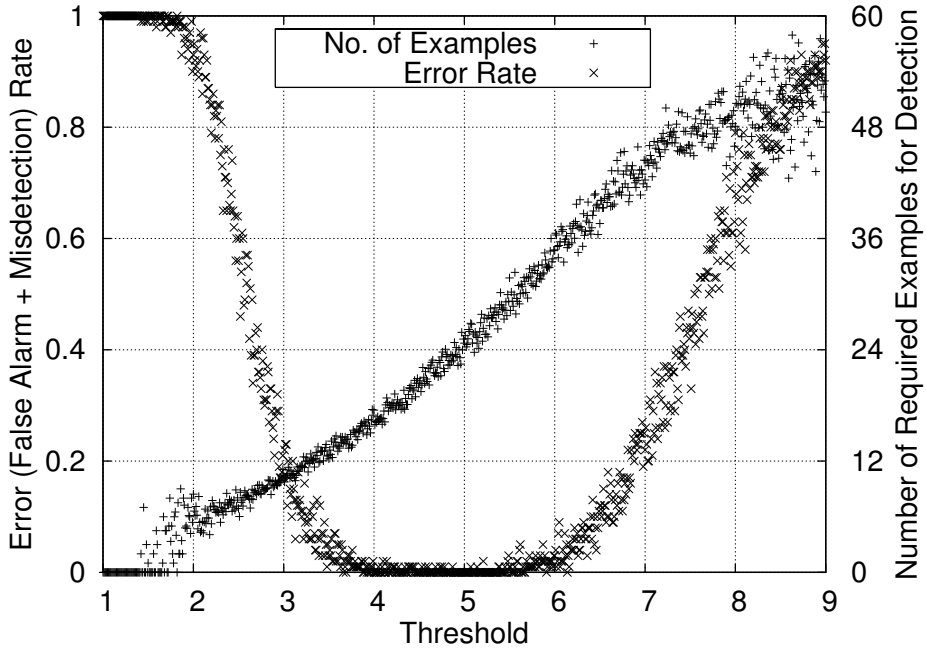

Fig. 5.13 Error (false alarm and misdetection) rates of and numbers of required examples for detecting the first change for LID on the first 1200 examples in the Moving Hyperplane problem. $V_{\rm th}$ (threshold) $\in [1, 9]$, $W = 100$, and $\gamma = 0.99$. Results were averaged over 100 trials.

changes, the parameter settings for LID were quite difficult. Setting parameters is a serious problem when using LID, and we have to set appropriate parameters before LID is presented with training examples.

5.4.3 Discussion and Future Work

The role of the threshold $V_{\rm th}$ is to adjust the sensitivity to changes in the LID model. A low threshold enables quick detection, but causes false alarms. In contrast, a high threshold enables robustness to noise, but causes misdetection. As shown in Figures 5.10 and 5.13, the best threshold is different for each learning problem. We cannot find the best threshold without prior knowledge of the problem. This prior knowledge includes the presence of noise, the difficulty of the learning problem, and the size of problem domain, respectively. Moreover, the best threshold is different depending on whether a user would like to control either false alarm or misdetection

(e.g. we have to reduce false alarms in spam filtering problems).

Adjusting the threshold automatically is necessary in order to use LID practically in the situation where there is less or no prior knowledge. We consider that humans are able to adjust parameters that correspond to their sensitivity to changes while learning sequentially, according to knowledge that humans get from their experiences. LID may be able to use the degree of confidence in and the recent accuracy of the online classifier as this knowledge. For example, if confident classifications are frequently rejected despite the fact that the converged value of accuracy is high and the target concept is not changing, then the threshold should be increased because there would be much noise. We will consider such a method for adjusting the threshold in our future work.

5.5 Summary

We first considered the significant difference between humans and machine learning systems, which treat any misclassification as the same misclassification. We therefore conducted a human behavior experiment, with a well-known pattern classification dataset, to investigate our working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. Consequently, we got the experimental results that suggest the validity of our working hypothesis, and we noticed that the successive rejection of highly confident classifications is significant for change detection.

We then thought the *l*eaky *i*ntegrate-and-*f*ire (LIF) model could be applied to change detection. In the LIF model, the membrane potential decays exponentially with a time constant toward a resting value until a new injected current arrives. Upon the arrival of the new injected current, the membrane potential is instantaneously increased. If the membrane potential crosses a threshold, a spike is formed and the membrane potential is reset to a resting value. We thus proposed the *l*eaky *i*ntegrate-

and-*d*etect (LID) model that regards the membrane potential increased by injected currents as a state value increased by misclassifications of a single online classifier. LID determines the increased value by using the degree of confidence in and the recent accuracy of the online classifier, according to our working hypothesis. LID can use any online classifier such as k-Nearest Neighbors and Naive Bayes. Experimental results showed LID was able to detect sudden and significant changes quickly and accurately in an environment that also includes noise and gradual changes. The results also showed the effectiveness of the input function that uses both the degree of confidence in and the recent accuracy of the online classifier.

However, it is necessary for LID to set three appropriate parameters, especially the threshold, before LID is presented with training examples. Our goal is to adjust the threshold automatically for the problem to be solved. Moreover, we would like to apply this knowledge about the degree of confidence from using LID to Todi and ACE in our future work.

Chapter 6

Conclusions

There are serious problems with online machine learning, especially in environments where the statistical properties of the target variable may change over time. This change is known as *concept drift*. An ideal online learning system for handling concept drift should be able to: (1) respond to both sudden and gradual changes; (2) recognize and treat recurring concepts effectively; (3) have both robustness to noise and sensitivity to true change; and (4) detect the occurrence of change. We have conducted interdisciplinary studies as stepping-stones to developing an online learning system that has these four abilities. In this chapter, we first present the contributions of this dissertation and then discuss future work.

6.1 Contributions of this Dissertation

First, we proposed a system that uses multiple classifiers, the *a*daptive *c*lassifiers-*e*nsemble (ACE) system. Using multiple classifiers would be a more natural solution to learning concept drift than continuously revising a single classifier. Therefore, several systems have been proposed that use ensembles of classifiers for learning concept drift [24, 25, 49, 50, 89–92, 96]. Most of these systems use a majority vote from the outputs of up-to-date base classifiers.

The ACE system forms its output by using a weighted majority vote from the outputs of a single online classifier that is continuously updated and many batch classifiers that are not updated. ACE uses confidence intervals for the recent predictive accuracies of batch classifiers to detect concept drift and builds a new batch classifier when concept drift is detected. ACE focuses especially on handling recurring concepts, and results showed that ACE was able to handle recurring concepts better than conventional systems. However, the original ACE system does not have any classifier pruning method, and its original weighting method does not sufficiently reduce interference from old classifiers.

We therefore added a new classifier pruning method and improved the original weighting method. To handle recurring concepts in the pruning method, ACE uses two sets of weights, one to decide the system output and the other to determine the order of pruning. The improved weighting method uses a confidence interval for the recent predictive accuracies of base classifiers and thus sufficiently reduces interference from old classifiers. Experimental results showed that the enhanced ACE system was better able to learn concept drift than the original ACE system.

Using both online and batch classifiers and a drift detection mechanism is a new approach to learning concept drift and enables ACE to respond both sudden and gradual changes and to handle recurring concepts well. This study therefore contributes to the development of abilities (1), (2), and (4). However, ACE could not detect concept drift accurately in real-world problems because the parameter settings are too difficult for ACE to use practically.

Next, we proposed a simpler method that more accurately detects concept drift that we call the STEPD method. A lot of work has been done on detecting changes in online data streams [8, 12, 14, 46, 63, 64], but most of it is based on estimating the underlying distribution of examples, which requires a large number of examples. Several methods of detecting concept drift in a small number of examples have been developed that monitor prediction errors of a single online classifier [3, 31]. These

methods assume that a significant increase in the error rate for the online classifier suggests that the target concept is changing. In contrast to methods that estimate the underlying distribution of examples, they also do not depend on the type of input attributes.

The STEPD method monitors the predictive accuracy of a single online classifier and detects significant decreases in the predictive accuracy by using a *s*tatistical *t*est of *e*qual *p*roportions to *d*etect concept drift. The online classifier is reinitialized to prepare for the learning of the next concept when concept drift is detected. Results showed that the statistical test enabled STEPD to quickly and accurately detect various types of concept drift in synthetic datasets. However, false alarms significantly worsen the predictive accuracy of STEPD because STEPD uses a single online classifier and thus performs poorly after the reinitialization of the online classifier.

Therefore, we proposed a system that can mitigate the bad influence of false alarms on predictive accuracy, the *t*wo *o*nline classifiers for learning and *d*etect*i*ng concept drift (Todi) system. When concept drift is detected by the STEPD method, one of the two classifiers is reinitialized, and the other one is not. Using two online classifiers also enables Todi to confirm the correctness of the last detection and to accurately notify a user of the occurrence of concept drift. To confirm the correctness, Todi compares the predictive accuracies of the two classifiers by using the same statistical test of equal proportions that STEPD uses. If the predictive accuracy of the reinitialized classifier is significantly better than that of the non-reinitialized classifier, Todi assumes that the last detection was correct. Experimental results showed that Todi performed well in learning and detecting concept drift in a synthetic dataset. We then used a spam filtering dataset to evaluate Todi and demonstrated that Todi, which uses two Bogofilters, performed better than the single Bogofilter (Bogofilter is a well-known and successful spam filter [83]).

Using two online classifiers is a new approach to learning and detecting concept drift and enables Todi to have both robustness to noise and sensitivity to true change.

This study therefore contributes to the development of abilities (3) and (4). However, Todi does not consider the occurrence of recurring concepts.

Finally, we proposed a method for detecting sudden concept drift that was inspired by human behavior, the *l*eaky *i*ntegrate-and-*d*etect (LID) model. We considered the significant difference between humans and machine learning systems, which treat any misclassification as the same misclassification. We therefore conducted a human behavior experiment to investigate the working hypothesis that humans can detect changes quickly when their confident classifications are rejected despite the fact that their recent classification accuracy is high. Results suggested the validity of the working hypothesis, and we noticed that the successive rejection of highly confident classifications is significant for change detection.

Based on this finding, we thought the *l*eaky *i*ntegrate-and-*f*ire (LIF) model, which is a spike generation model [48, 56], could be applied to change detection. In the LIF model, the membrane potential decays exponentially with a time constant toward a resting value until a new injected current arrives. Upon the arrival of the new injected current, the membrane potential is instantaneously increases. If the membrane potential crosses a threshold, a spike is formed and the membrane potential is reset to a resting value. We thus proposed the LID model that regards the membrane potential increased by injected currents as a state value increased by the misclassifications of a single online classifier. LID determines the incremental value by using the recent accuracy of and the degree of confidence in the online classifier. Computer experiments showed that LID enabled the online classifier to respond quickly and accurately to sudden and significant changes in an environment that also includes noise and gradual changes. The integration of the state value with time decay reduces false alarms, which are caused by noise and gradual changes, and enables LID to detect frequent occurrences of misclassifications, which are caused by sudden and significant changes.

Using the degree of confidence is a new approach to detecting concept drift that enables LID to detect sudden and significant changes quickly. This study therefore

contributes to the development of ability (4). However, the parameter settings are also too difficult for LID to use practically.

To summarize, we have conducted interdisciplinary studies as stepping-stones to achieving an effective online learning system for handling concept drift, and this dissertation has contributed to the development of the four abilities that an ideal system should have.

6.2 Future Work

We first plan to combine Todi with ACE to create a more intelligent online learning system for handling concept drift. The combined system would consist of two online classifiers and many batch classifiers. Our goal is to take advantage of Todi's detection accuracy to better add and remove batch classifiers. We would also like to use the degree of confidence in base classifiers for weighting the outputs of the classifiers and detecting concept drift.

Next, we will investigate the problem of adjusting parameters. Most learning systems have some or many parameters that users need to set appropriately for the problem to be solved. Users can easily find appropriate parameters for batch learning systems because they have all the training examples and thus understand the characteristics of the problem before using the training examples. They are also able to find the best parameters by trial and error. However, users find it difficult to set parameters appropriately for online learning systems, for which the training examples are given sequentially. If users do not have any prior knowledge of the problem, setting appropriate parameters is the most serious challenge. Furthermore, in a concept drift environment, the appropriate parameters may also change over time. Therefore, an ideal online learning system for handling concept drift should be also able to adjust parameters while the system is learning sequentially. Several systems that are able to adjust the size of the sliding window that they use have been proposed [47, 58, 99].

However, none of our proposed systems are able to adjust any parameters. Moreover, the parameters of ACE and of LID depend strongly on the problem to be solved. We would like to solve this parameter adjustment problem.

After that, we plan to work on acquiring *meta-knowledge* of concept drift for the problem to be solved. Real-world problems contain not only irregular changes but also regular changes that can be anticipated. For instance, spring comes at a similar time every year, and summer always follows spring. Understanding the cycle and sequence of change, which we call meta-knowledge, enables an online learning system to predict the next occurrence of concept drift. An ideal online learning system should be able to understand the meta-knowledge of concept drift for the problem to be solved. We plan to deal with the detection of concept drift in order to understand the cycle of change and to deal with the clustering of past classifiers so that we can recognize the current concept correctly. Measuring ensemble diversity is one way of clustering classifiers [54, 94, 104], and redundant classifiers can be removed by clustering.

Finally, we plan to continue studying *human learning*. Machine learning, with its enormous memory capacity and high performance CPUs, does not completely outperform human learning, especially in concept drift environments. It is also important that humans be able to adjust parameters while learning sequentially and to understand the meta-knowledge of concept drift for the problem to be solved. Human learning should therefore be better understood. The several relevant research areas in cognitive science were summed up by Widmer and Kubat [99]. Our challenge is to enable computers outperform humans.

In conclusion, we believe that an ideal online learning system for handling concept drift should be able to: (1) respond to both sudden and gradual changes; (2) recognize and treat recurring concepts effectively; (3) have both robustness to noise and sensitivity to true change; (4) detect the occurrence of change; (5) adjust parameters while the system is learning sequentially; and (6) understand the meta-knowledge of concept drift for the problem to be solved. Our ultimate goal is to create an online learning

system that has all six abilities.

Bibliography

[1] A. Agresti, *Categorical Data Analysis*, 2nd ed. Wiley-Interscience, 2002.

[2] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Lerning*, vol. 6, no. 1, pp. 37–66, 1991.

[3] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, "Early drift detection method," in *Proceedings of the ECML/PKDD 2006 Workshop on Knowledge Discovery from Data Streams*, 2006, pp. 77–86.

[4] P. L. Bartlett, S. Ben-David, and S. R. Kulkarni, "Learning changing concepts by exploiting the structure of change," *Machine Lerning*, vol. 41, no. 2, pp. 153–174, 2000.

[5] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, 1993.

[6] J. L. Bentley, "Multidimensional binary search trees used for associative," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[7] M. Birgmeier, "A fully kalman-trained radial basis function network for nonlinear speech modeling," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995, pp. 259–264.

[8] C. M. Bishop, "Novelty detection and neural network validation," *IEE Proceedings: Vision, Image and Signal Processing*, vol. 141, no. 4, pp. 217–222, 1994.

[9] L. Breiman, "Bagging predictors," *Machine Lerning*, vol. 24, no. 2, pp. 123–140, 1996.

[10] L. Breiman, "Arcing classifiers," *The Annals of Statistics*, vol. 26, no. 3, pp. 801–849, 1998.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regressioon Trees*. Champman & Hall, 1984.

[12] B. E. Brodsky and B. S. Darkhovsky, *Non-Parametric Methods in Change-Point Problems*. Kluwer Academic, 1993.

[13] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[14] J. Chen and A. K. Gupta, Eds., *Parametric Statistical Change Point Analysis*. Birkhauser, 2000.

[15] G. V. Cormack, "Trec 2006 spam track overview," in *Proceedings of the 15th Text Retrieval Conference*, 2006.

[16] G. V. Cormack, "TREC 2006 spam track public corpora," 2006. [Online]. Available: http://plg.uwaterloo.ca/~gvcormac/treccorpus06/

[17] G. V. Cormack and T. R. Lynam, "Trec 2005 spam track overview," in *Proceedings of the 14th Text Retrieval Conference*, 2005.

[18] G. V. Cormack and T. R. Lynam, "TREC 2005 spam track public corpus," 2005. [Online]. Available: http://plg.uwaterloo.ca/~gvcormac/treccorpus/

[19] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. IT-13, no. 1, 1967.

[20] B. V. Dasarathy, Ed., *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society, 1991.

[21] B. V. Dasarathy, J. S. Sanchez, and S. Townsend, "Nearest neighbour editing and condensing tools – synergy exploitation," *Pattern Analysis & Applications*, vol. 3, pp. 19–30, 2000.

[22] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle, "A case-based technique for tracking concept drift in spam filtering," *Knowledge-Based System*, vol. 18, no. 4-5, pp. 187–195, 2005.

[23] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley-Interscience, 2000.

[24] W. Fan, "StreamMiner: a classifier ensemble-based engine to mine concept-drifting data streams," in *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004, pp. 1257–1260.

[25] W. Fan, "Systematic data selection to mine concept-drifting data streams," in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 128–137.

[26] T. Fawcett, " 'in vivo' spam filtering: a challenge problem for data mining," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 2, pp. 140–148, 2003.

[27] R. A. Fisher, *Statistical Methods for Research Workers*, 14th ed. Edinburgh: Oliver and Boyd, 1970.

[28] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[29] Y. Freund and R. E. Schapire, "A short introduction to boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[30] J. H. Friedman, J. L. Bentley, and R. A. Frinkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[31] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*, 2004, pp. 285–295.

[32] P. Graham, "A plan for spam," 2002. [Online]. Available: http://www.paulgraham.com/spam.html

[33] P. Graham, "Better bayesian filtering," 2004. [Online]. Available: http://www.paulgraham.com/better.html

[34] M. Harries, "Splice-2 comparative evaluation: Electricity pricing," School of

Computer Science and Engineering, The University of South Wales, Tech. Rep. UNSW-CSE-TR-9905, 1999.

[35] M. B. Harries, C. Sammut, and K. Horn, "Extracting hidden context," *Machine Lerning*, vol. 32, no. 2, pp. 101–126, 1998.

[36] S. Haykin, *Adaptive Filter Theory*, 4th ed. Prentice Hall, 2001.

[37] D. P. Helmbold and P. M. Long, "Tracking drifting concepts by minimizing disagreements," *Machine Lerning*, vol. 14, no. 1, pp. 27–45, 1994.

[38] R. V. Hogg and E. A. Tanis, *Probability and Statistical Inference*, 5th ed. Prentice Hall, 1997.

[39] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning rbf (GAP-RBF) networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 6, 2004.

[40] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning rbf (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, 2005.

[41] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 97–106.

[42] T. Joachims, "Making large-scale svm learning practical," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. J. Smola, Eds. MIT Press, 1998, pp. 169–184.

[43] T. Joachims, "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 217–226.

[44] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 338–345.

[45] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to se-

quential learning with neural networks," *Neural Computation*, vol. 5, no. 6, pp. 954–975, 1993.

[46] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proceedings of the 30th Very Large Data Base Conference*, 2004, pp. 180–191.

[47] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281–300, 2004.

[48] B. W. Knight, "Dynamics of encoding in a population of neurons," *Journal of General Physiology*, vol. 59, pp. 734–766, 1972.

[49] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: a new ensemble method for tracking concept drift," in *Proceedings of the 3rd International IEEE Conference on Data Mining*, 2003, pp. 123–130.

[50] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *Proceedings of the 22nd International Conference on Machine Learninig*, 2005, pp. 449–456.

[51] M. Kubat and G. Widmer, "Adapting to drift in continuous domain." in *Proceedings of the 8th European Conference on Machine Learning*, 1995, pp. 307–310.

[52] A. Kuh, T. Petsche, and R. L. Rivest, "Learning timevarying concepts," in *Advances in Neural Information Processing Systems*, vol. 3, 1991, pp. 183–189.

[53] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Proceedings of the 5th International Workshop on Multiple Classifier Systems*, 2004, pp. 1–15.

[54] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, no. 2, pp. 181–207, 2003.

[55] M. H. Kutner, C. J. Nachtsheim, J. Neter, and W. Li, *Applied Linear Statistical Models*, 5th ed. McGraw-Hill/Irwin, 2004.

[56] L. Lapicque, "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarization," *Journal de Physiologie et de Pathologie Generalej*, vol. 9, pp. 620–635, 1907.

[57] P. S. Laplace, *Théorie analytique des probabilitités*. Courcier, 1812.

[58] M. M. Lazarescu, S. Venkatesh, and H. H. Bui, "Using multiple windows to track concept drift," *Intelligent Data Analysis*, vol. 8, no. 1, pp. 29–59, 2004.

[59] J. R. Lewis and J. Sauro, "When 100% really isn't 100%: Improving the accuracy of small-sample estimates of completion rates," *Journal of Usability Studies*, vol. 1, no. 3, pp. 136–150, 2006.

[60] Y. Li, N. Sundararajan, and P. Saratchandran, "Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems," *IEE Proceedings: Control Theory Application*, vol. 147, no. 4, pp. 476–484, 2000.

[61] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212–261, 1994.

[62] P. M. Long, "The complexity of learning according to two models of a drifting environment," *Machine Lerning*, vol. 37, no. 3, pp. 337–354, 1999.

[63] M. Markou and S. Singh, "Novelty detection: a review — part 1: Statistical approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[64] M. Markou and S. Singh, "Novelty detection: a review — part 2: Neural network based approaches," *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, 2003.

[65] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," in *Advanced Lectures on Machine Lerning*, ser. Lecture Notes in Artificial Intelligence, vol. 2600, 2003, pp. 119–184.

[66] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes – which naive bayes?" in *Proceedings of the 3rd Conference on Email and Anti-Spam*, 2006.

[67] E. W. Minium and B. M. King, *Statistical Reasoning in Psychology and Education*, 4th ed. John Wiley & Sons, 2002.

[68] T. M. Mitchell, *Machine learning*. McGraw-Hill, 1997.

[69] K. Nishida, S. Shimada, S. Ishikawa, and K. Yamauchi, "Detecting sudden and

significant concept drift with a method inspired by human behavior," *IEICE Transactions on Information and Systems (Japanese Edition)*, vol. J91-D, no. 1, pp. 51–64, 2008, in Japanese.

[70] K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for concept-drifting environments," IEICE, Tech. Rep. NC2005-98, 2006, in Japanese.

[71] K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for changing situations," IEICE, Tech. Rep. PRMU2006-272, 2007.

[72] K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for tracking concept drift," in *Proceedings of the 6th International Conference on Machine Learning and Cybernetics*, 2007, pp. 3607–3612.

[73] K. Nishida and K. Yamauchi, "Detecting concept drift using statistical testing," in *Proceedings of the 10th International Conference on Discovery Science*, ser. Lecture Notes in Artificial Inteligence, vol. 4755, 2007, pp. 264–269.

[74] K. Nishida, K. Yamauchi, and T. Omori, "An on-line learning algorithm with dimension selection using minimal hyper basis function networks," in *Proceedings of the 11th International Conference on Neural Information Processing*, ser. Lecture Notes in Computer Science, vol. 3316, 2004, pp. 502–507.

[75] K. Nishida, K. Yamauchi, and T. Omori, "ACE: Adaptive classifiers-ensemble system for concept-drifting environments," in *Proceedings of the 6th International Workshop on Multiple Classifier Systems*, ser. Lecture Notes in Computer Science, vol. 3541, 2005, pp. 176–185.

[76] K. Nishida, K. Yamauchi, and T. Omori, "Adaptive classifiers-ensemble system for concept-drifting environments," in *Proceedings of the 2006 Annual Conference of the Japanese Neural Network Society*, 2006, pp. 138–139, in Japanese.

[77] K. Nishida, K. Yamauchi, and T. Omori, "An online learning algorithm with dimension selection using minimal hyper basis function networks," *Systems and Computers in Japan*, vol. 37, no. 11, pp. 176–185, 2006, (translated from *IEICE Transactions on Information and Systems, PT.2 (Japanese Edition)*, vol. J88-

D-II, no. 3, pp. 459-468, 2005).

[78] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[79] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. J. Smola, Eds. MIT Press, 1998, pp. 185–208.

[80] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, 1990.

[81] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[82] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[83] E. S. Raymond, D. Relson, M. Andree, and G. Louis, *Bogofilter*, v1.1.5, 2007. [Online]. Available: http://bogofilter.sourceforge.net/

[84] G. Robinson, "Spam detection," 2002. [Online]. Available: http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html

[85] G. Robinson, "A statistical approach to the spam problem," *Linux journal*, no. 107, 2003. [Online]. Available: http://www.linuxjournal.com/article/6467

[86] R. E. Schapire, "The boosting approach to machine learning: an overview," in *Nonlinear Estimation and Classification*. Springer, 2003.

[87] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine Lerning*, vol. 1, no. 3, pp. 317–354, 1986.

[88] K.-M. Schneider, "On word frequency information and negative evidence in naive bayes text classification," in *Proceedings of the 4th International Conference on Advances in Natural Language Processing*, 2004, pp. 474–485.

[89] M. Scholz and R. Klinkenberg, "An ensemble classifier for drifting concepts," in *Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Stream*, 2005, pp. 53–64.

[90] M. Scholz and R. Klinkenberg, "Boosting classifiers for drifting concepts," *Intel-*

ligent Data Analysis, vol. 11, no. 1, pp. 3–28, 2007.

[91] K. O. Stanley, "Learning concept drift with a committee of decision trees," Department of Computer Sciences, University of Texas at Austin, Tech. Rep. AI-03-302, 2003.

[92] W. N. Street and Y. S. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 377–382.

[93] A. Tsymbal, "The problem of concept drift: definitions and related work," Department of Computer Science, Trinity College Dublin, Tech. Rep. TCD-CS-2004-15, 2004.

[94] A. Tsymbal, M. Pechenizkiy, and P. Cunningham, "Diversity in search strategies for ensemble feature selection," *Information Fusion*, vol. 6, no. 1, pp. 83–98, 2005.

[95] V. N. Vapnik, *Statistical learning theory*. John Wiley & Sons, 1998.

[96] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 226–235.

[97] G. Widmer, "Tracking context changes through meta-learning," *Machine Learning*, vol. 27, no. 3, pp. 259–286, 1997.

[98] G. Widmer and M. Kubat, "Effective learning in dynamic environments by explicit concept tracking," in *Proceedings of the 6th European Conference on Machine Learning*, 1993, pp. 227–243.

[99] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[100] D. H. Widyantoro and J. Yen, "Relevant data expansion for learning concept drift from sparsely labeled data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 401–412, 2005.

[101] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, 2005.

[102] F. Yates, "Contingency table involving small numbers and the χ^2 test," *Journal of the Royal Statistical Society Supplement*, vol. 1, pp. 217–235, 1934.

[103] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation*, vol. 9, no. 2, pp. 461–478, 1997.

[104] I. Žliobaitė, "Introduction of new expert and old expert retirement in ensemble learning under drifting concept," in *Progress in Pattern Recognition*, ser. Advances in Pattern Recognition, S. Singh and M. Singh, Eds., 2007, pp. 64–74.

Publications

Journal Papers

(1) K. Nishida, S. Shimada, S. Ishikawa, and K. Yamauchi, "Detecting sudden and significant concept drift with a method inspired by human behavior," *IEICE Transactions on Information and Systems (Japanese Edition)*, vol. J91-D, no. 1, pp. 51-64, 2008, in Japanese.

(2) K. Nishida, K. Yamauchi, and T. Omori, "An online learning algorithm with dimension selection using minimal hyper basis function networks," *Systems and Computers in Japan*, vol. 37, no. 11, pp. 176–185, 2006, (translated from *IEICE Transactions on Information and Systems, PT.2 (Japanese Edition)*, vol. J88-D-II, no. 3, pp. 459-468, 2005).

International Conference Papers

(4) K. Nishida and K. Yamauchi, "Detecting concept drift using statistical testing," in *Proceedings of the 10th International Conference on Discovery Science*, ser. Lecture Notes in Artificial Inteligence, vol. 4755, 2007, pp. 264–269.

(5) K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for tracking concept drift," in *Proceedings of the 6th International Conference on Machine Learning and Cybernetics*, 2007, pp. 3607–3612.

(6) K. Nishida, K. Yamauchi, and T. Omori, "ACE: Adaptive classifiers-ensemble system for concept-drifting environments," in *Proceedings of the 6th International Workshop on Multiple Classifier Systems*, ser. Lecture Notes in Computer Science, vol. 3541, 2005, pp. 176–185.

(7) K. Nishida, K. Yamauchi, and T. Omori, "An on-line learning algorithm with dimension selection using minimal hyper basis function networks," in *Proceedings of the 11th International Conference on Neural Information Processing*, ser. Lecture Notes in Computer Science, vol. 3316, 2004, pp. 502–507.

(8) K. Nishida, K. Yamauchi, and T. Omori, "An on-line learning algorithm with dimension selection using minimal hyper basis function networks," in *Proceedings of the SICE Annnual Conference 2004*, 2004, pp. 2610–2615.

Domestic Conference Papers and Technical Reports

(9) K. Nishida, S. Shimada, S. Ishikawa, and K. Yamauchi, "Detecting sudden and significant concept drift," in *2007 Joint Convention Record of the Hokkaido Chapters of the Institutes of Electrical and Information engineers*, 2007, pp. 305–306, in Japanese.

(10) K. Nishida and K. Yamauchi, "Adaptive classifiers-ensemble system for changing situations," IEICE Technical Report, PRMU2006-272, 2007, pp. 103-108.

(11) K. Narumi, K. Nishida, and K. Yamauchi, "An online spam filtering system for changing situations using multiple classifiers," IEICE Technical Report, PRMU2006-235, 2007, in Japanese.

(12) K. Nishida, K. Yamauchi, and T. Omori, "Adaptive classifiers-ensemble system for concept-drifting environments," in *Proceedings of the 2006 Annual Conference of the Japanese Neural Network Society*, 2006, pp. 138–139, in Japanese.

(13) K. Nishida, K. Yamauchi, and T. Omori, "Adaptive classifiers-ensemble system

for concept-drifting environments," IEICE, Tech. Rep. NC2005-98, 2006, in Japanese.

(14) K. Nishida, K. Yamauchi, and T. Omori, "An on-line learning algorithm with dimension selection using minimal hyper basis function networks," IEICE, Tech. Rep. NC2003-215, vol. 103, no. 734, 2004, pp. 127–132, in Japanese.

(15) K. Nishida, K. Yamauchi, and T. Omori, "A sequential learning algorithm using minimal hyper basis function networks," in *Proceedings of the Sixth Workshop on Information-Based Induction Sciences*, 2003, pp. 211–216, in Japanese.

(16) K. Nishida and K. Yamauchi, "A sequential learning algorithm using minimal hyper basis function networks," in *Proceedings of the 2003 Annual Conference of the Japanese Neural Network Society*, 2003, pp. 226–227, in Japanese.