A MapReduce-Based Ensemble Learning Method with Multiple Classifier Types and Diversity for Condition-based Maintenance with Concept Drifts

Chun-Cheng Lin, Lei Shu, Der-Jiunn Deng, Tzu-Lei Yeh, Yu-Hsiang Chen, and Hsin-Lung Hsieh*

***Abstract*—Condition-based maintenance (CBM) in Industry 4.0 collects a huge amount of production data stream continuously from IoT devices attached to machines to forecast the time when to maintain machines or replace components. However, as conditions of machines change dynamically with time owing to machine aging, malfunction or replacement, the concept of capturing the forecasting pattern from the data stream could drift unpredictably so that it is hard to find a robust forecasting method with high precision. Therefore, this work proposes an ensemble learning method with multiple classifier types and diversity for CBM in manufacturing industries, to address the bias problem when using only one base classifier type. Aside from manipulating data diversity, this method includes multiple classifier types, dynamic weight adjusting, and data-based adaption to concept drifts for offline learning models, to promote precision of the forecasting model and precisely detect and adapt to concept drifts. With these features, the proposed method requires powerful computing resources to efficiently respond to practical CBM applications. Therefore, furthermore, the implementation of this method based on the MapReduce framework is proposed to increase computational efficiency. Simulation results show that this method can detect and adapt to all concept drifts with a high precision rate.**

***Index Terms*—Concept drift, ensemble learning, MapReduce, condition-based maintenance, Industry 4.0**

I. Introduction

Data classification has played a crucial role in big data analysis, and has a variety of applications, e.g., spam filtering, medical diagnosis, and fault detection and classification (FDC) in semiconductor manufacturing. A recent trend on data classification focuses on how to detect and adapt to the problem of concept drifts. This work focuses on condition-based maintenance (CBM) with concept drifts in Industry 4.0, in which the enormous historical data stream of conditions of machines is collected continuously from sensors or IoT devices attached to machines [1], and data classification algorithms are adopted to analyze the data to detect the time when some components in machines start to perform abnormally and should be replaced in advance, to avoid machines from crashing or manufacturing a huge amount of defective products [2]. Hence, historical data is the main factor to affect the precision performance of data classification algorithms. However, as time goes by, machine components become aging, malfunctioned, or need to be maintained, so that the status of the machine has kept changing dynamically. Therefore, the concept of capturing the classification patterns may drift unpredictably.

Lots of works addressed the problem of concept drifts by ensemble learning methods, which are of supervised learning in machine learning. Ensemble learning first establishes a number of classifiers, and then generates an ensemble results by applying some voting scheme to aggregate the results generated by all classifiers. For instance, Minku and Yao [3] proposed a promising ensemble learning method called Diversity for Dealing with Drift (DDD), which manipulates data diversity to generate ensemble models with different degrees of diversity, in which each ensemble model consists of a number of base classifiers of the same type, to adapt to the concept drift problem.

A distributed framework coordinates networked computing devices to achieve complex tasks, e.g., leak-point prediction from big data [4][5]. MapReduce is a distributed cloud computing framework that has received a lot of attention for big data processing [6]. For instance, Palit and Reddy [7] proposed a boosted ensemble classifier that facilitates simultaneous participation of multiple computing resources, and further applied the MapReduce to increase the computational efficiency of an ensemble learning method. The idea of MapReduce is based on a divide-and-conquer strategy, which divides the data into a number of smaller data pieces, then adopts the Map function to process each data piece in parallel to obtain an immediate result, and then adopts the Reduce function to aggregate these immediate results into a

The work was supported partially by Industrial Technology Research Institute, Taiwan, under the project of Big Data Technologies and Applications.

C.-C. Lin, T.-L. Yeh, and Y.-H. Chen are with Department of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan. E-mails: cclin321@nctu.edu.tw, petery3xo3@hotmail.com, k5527531@yahoo.com.tw

L. Shu is with University of Lincoln, Lincoln, UK and Nanjing Agricultural University, China. E-mail: lei.shu@ieee.org

D.-J. Deng is with Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua 500, Taiwan. E-mail: djdeng@cc.ncue.edu.tw

H.-L. Hsieh is with Computational Intelligence Technology Center, Industrial Technology Research Institute, Hsinchu 310, Taiwan. E-mail: hlhsieh77@gmail.com

* D.-J. Deng is the corresponding author of this article.

final result. Note that the Map and Reduce functions can be customized by users, and hence can be applied to various algorithms.

This work proposes a multiple-classifier-type DDD (MDDD for short) ensemble learning method for CBM in manufacturing industries, which extends the DDD with multiple classifier types, and includes a novel data-based adaption scheme that allows offline base classifiers, to increase the precision of the forecasting model. In general, however, ensemble learning methods include complex designs and time-consuming classifiers, so that they often take too much computational time in some cases owing to too many data dimensions, enormous amount of training data, or too many base learners in the ensemble model. Therefore, the implementation of the proposed MDDD based on the MapReduce framework is proposed, to increase the computational efficiency to meet practical computing requirements of CBM applications. Through simulation on a benchmarking dataset with concept drifts, the performance of the proposed MDDD is verified.

II. Related Works

Given a dataset $D = \{e_1, e_2, \ldots, e_i, \ldots\}$, in which e_i represents a data point; $e_i = (X_i, y_i)$ in which X_i is a feature vector and y_i is a class label; each element in feature vector X_i is called an attribute. Given a data point $e_i = (X_i, y_i)$, the process of mapping feature vector X_i to class label y_i through some function f is called classification (i.e., $y \leftarrow f(X_i)$), and the function f is called a classifier.

Most data classification applications in practice are designed for data stream, e.g., IoT devices attached to manufacturing machines continuously collects production data on machine conditions. A concept represents the whole distribution of the classification problem at some specific time point, which is characterized by a joint distribution $P(X, y)$ in which X includes the input feature vectors, and y represents their class labels. A concept drift means a change of the distribution [3].

Concept drifts include four patterns [8]: abrupt drift, gradual drift, incremental drift, and reoccurring drift. For example, consider a two-dimensional feature space, including a circle A centered at (0.5, 0.5) with radius 0.2 and a circle B centered at (0.5, 0.5) with radius 0.5. Define the feature vectors X within circle A to be labeled by $y = -1$, and those outside circle A to be labeled by $y = 1$. Therefore, the joint distribution P of the feature vectors X with class label $y = -1$ constitutes a concept. If the joint distribution P changes from circle A to circle B as time goes by, the process is an abrupt concept drift.

In practice, concepts are unstable and changes dynamically and unpredictably with time, e.g., in manufacturing industries, concepts for the data of machine conditions may change when machines become aging, malfunctioned, or replaced. Ensemble learning methods have been shown to perform better than single learning models [9]. In ensemble learning, a number of "experts" constitute a "committee" that applies some voting scheme to aggregate a final consistent result among the immediate results forecasted by all experts.

Recently, a lot of ensemble learning methods for addressing concept drifts have been proposed. For instance, Minku et al. [10] employed an online bagging strategy to establish an online ensemble learning model. Freund and Schapire [11] addressed concept drifts by an incremental learning ensemble model that integrates a dynamically weighted majority voting scheme. Minku and Yao [3] manipulated data diversity to establish an online ensemble learning model called DDD to address concept drifts. Gama et al. [8] surveyed a lot of methods for concept drift adaptation.

The DDD is an online ensemble learning method [3] consists of three stages: ensemble, detection, and adaptation. At Stage 1, two ensemble models with high and low diversities, respectively, are established through manipulating the training dataset. At Stage 2, the early drift detection method (EDDM) [12] is adopted to detect whether a concept drift occurs. If a concept drift is detected, the DDD enters Stage 3. At Stage 3, the recent training data points are adopted to train two new ensemble models with high and low diversities. Then, new and old ensemble models with high and low diversities are coordinated to adapt to the concept drift. One of the most significant contributions of the DDD is that the DDD discovered different behaviors of ensembles trained by different degrees of diversity under the presence of concept drifts, and used them as a stopgap measure to deal with concept drifts. Additionally, it developed a brilliant way to manipulate the diversity of ensembles by merely changing one parameter so that it can adapt to concept drifts easily. Simulation results also showed that DDD was very successful in dealing with concept drifts. However, the drawback of the DDD is that it can only be implemented by certain type of base classifier and only one type at a time. This leaves a problem: it is unknown which type of base classifier should be chosen. And, it is the motivation of this work to make up this deficiency.

III. Proposed Ensemble Method

All the base classifiers in the original DDD ensemble method are of the same type and can only be trained by online learning. The DDD did not consider diversity of multiple classifier types, and hence may have the bias problem of base classifiers. Bias problem is the weakness of the ensemble methods based on a single base classifier type, which may perform not well in some concepts due to the nature of this base classifier type. Although ensemble methods had been designed to address this kind of problem, the bias problem still existed when DDD uses only online bagging to establish ensembles based on one single classifier type. This work discovers this problem while testing the same dataset by several different base classifier types (also observable in Subsection IV-C). The same base classifier type always performs better for a certain certain concept, but may not for other concepts. Therefore, the precision of DDD after a concept drift may become worse, regardless of applying any base classifier type. As a result, the MDDD extends the DDD with multiple classifier types, and dynamically adjusts the

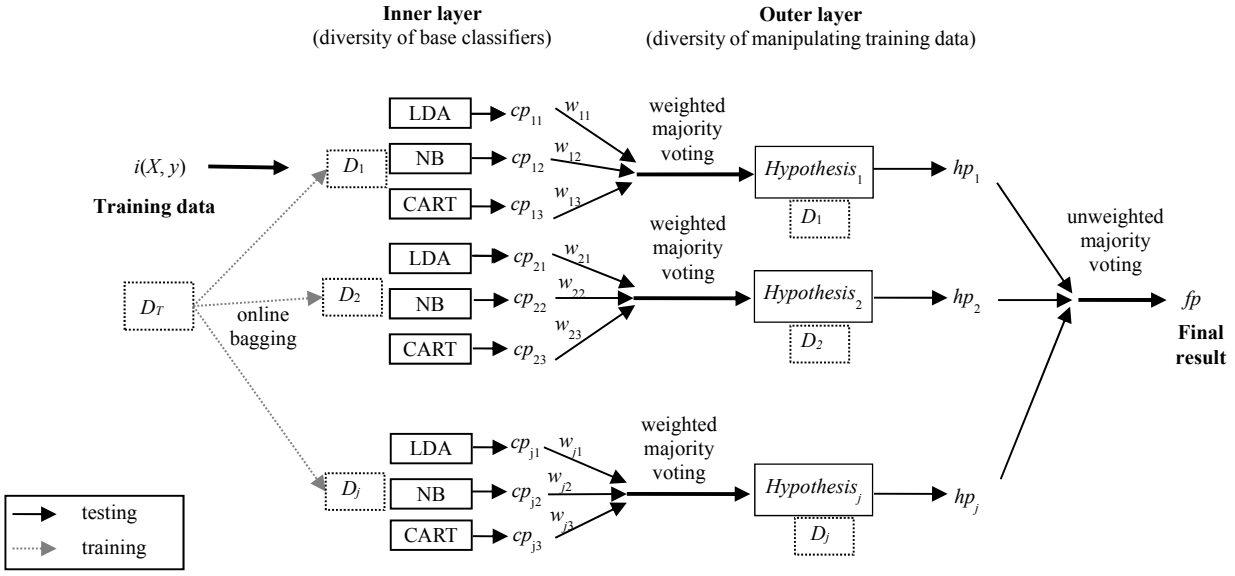

Fig 1. Illustration of Stage 1 in the MDDD.

weight of each classifier to adapt to the current concept. In addition, a lot of classifiers with good performance cannot be trained by online learning, but can be by offline learning. Therefore, different from the DDD, the MDDD allows offline classifier types. For offline learning base classifiers, the MDDD temporarily stores some necessary data before concept drifts, and then trains these offline learning classifiers when required to be retrained.

By doing so, the MDDD improves Stages 1 and 3 of the DDD, and proposed an implementation of the MapReduce framework. Next, this section introduces the three stages of the proposed MDDD, and the MapReduce framework.

A. Stage 1: Ensemble learning

The basic idea of a general ensemble learning method is as follows. First, j classifiers are established. Then, for each input data point, the j classifiers generate j results, and a certain voting scheme is adopted to vote the j results to generate a final result. However, it is not easy to establish j different classifiers. Hence, most pervious works only considered one base classifier (e.g., naïve Bayes (NB), support vector machine, and decision tree), and manipulated the training data so that j same-type classifiers with the same contents are trained to become j same-type classifiers with different contents. That is, these works were based on diversity of manipulating training data to avoid problems of variance and overfitting. However, such a strategy cannot solve the bias problem of using only one base classifier type.

To solve the above problem, the proposed MDDD considers ensemble learning with multiple different-type classifiers. The MDDD consists of outer and inner layers (Fig. 1), in which the inner layer is based on diversity of base classifiers to avoid the bias problem of classifiers; and the outer layer is based on diversity of manipulating training data to avoid the problem of variance and overfitting.

The diversity of manipulating training data in the outer layer of MDDD is achieved by online bagging [14]. Given a training dataset D_T, the outer layer applies online bagging to construct j different training datasets $D_1, D_2, \ldots, D_j$. That is, for each data point in D_T, each of the j training datasets randomly includes K copies of this data point in which K is a random variable following the Poisson distribution of a given parameter λ.

In the inner layer, each of the j training datasets $D_1, D_2, \ldots, D_j$ is further used to train r base classifiers of different types which may be trained by online and offline learning (depending on the user's settings), e.g., linear discriminant analysis (LDA), NB, and classification and regression tree (CART) in Fig. 1. And, for each of the j training datasets, the r base classifiers constitutes a hypothesis. Finally, the j hypotheses conclude a final result fp based on unweighted majority voting (Fig. 1). The final result fp applies unweighted majority voting because it is an aggregation of the outer layer, which is diversity of training data, which were generated randomly and are viewed as equal. Different from the aggregation of the inner layer, base classifiers have differences in nature.

To manifest the base classifiers with better precision performance in each hypothesis, the voting scheme in the inner layer follows the following weighted majority voting strategy. Consider adjusting the weights of r base classifiers in a hypothesis. Each weight is initialized with a value $1/r$. Every time when a base classifier generates a result, the result is compared with the actual result. If the two results are the same, the weight is multiplied with γ; otherwise, it is divided by γ. Finally, all weights in the same hypothesis are normalized for later use.

When initializing the MDDD (i.e., at the training stage), online bagging is adopted on the training dataset D_T to generate two major datasets called old-low-diversity and old-high-diversity, respectively, in the K variables in the

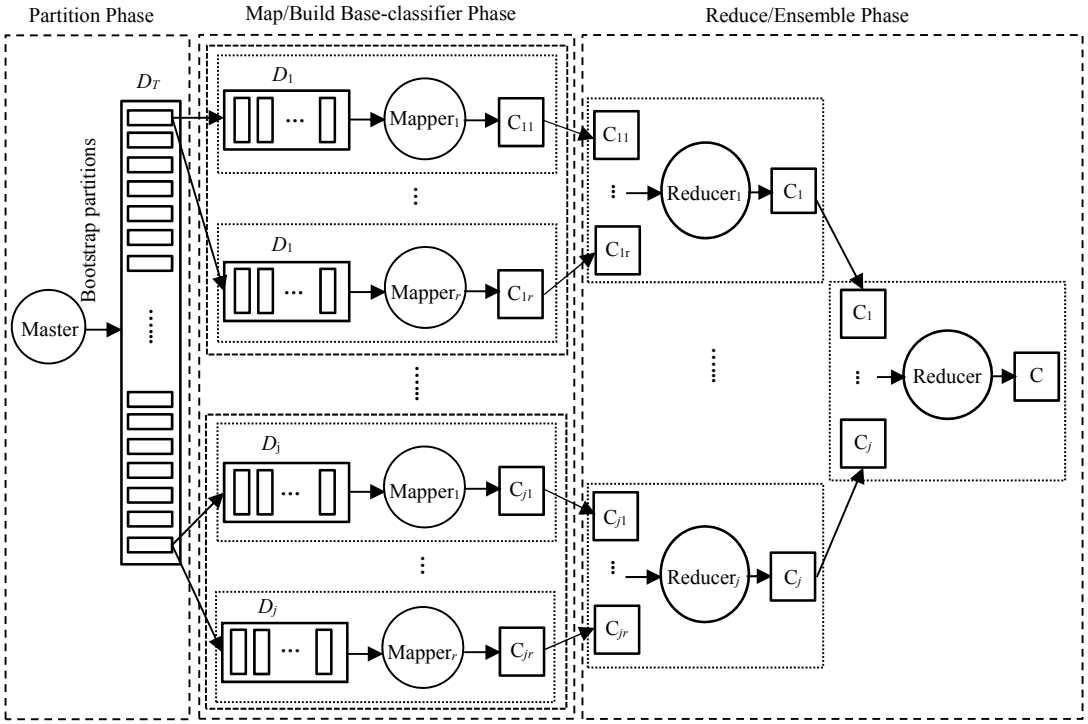

Fig 2. The MDDD based on the MapReduce framework.

old-low-diversity (resp., old-high-diversity) major dataset follows a Poisson distribution with $\lambda = 1$ (resp., $\lambda = 0.001$). Note that each major dataset consists of j training datasets, and the diversity represents the difference degree of the j training datasets generated by online bagging. When initializing the MDDD, the j training datasets in the old-low-diversity major dataset are adopted to train the initial r classifiers in jth hypothesis, but the old-high-diversity major dataset is not used until adaption at Stage 3.

After training the MDDD, consider the testing stage, i.e., each data point in the testing data is tested by the j hypotheses. If the class label of this testing data point forecasted by the MDDD is the same with the real class label, then the MDDD is recognized to obtain a correct forest; otherwise, a forecast fault. Then, the precision performance of the MDDD can be computed.

Different from the DDD that uses each new testing data point to train a number of base classifiers of the same type (e.g., NB classifier) at Stage 1, the MDDD does not. But, the MDDD stores the new data point only when a warning level is determined at Stage 2 (which will be introduced later), and waits for a drift level to be determined at Sage 2 to conduct online bagging based on these data points to retrain the j hypotheses.

B. Stage 2: Concept drift detection

After a testing result is obtained at Stage 1, Stage 2 is to detect whether a concept drift exists after the input data point is considered. The same with the DDD, this work applies the EDDM [12] to detect concept drifts. The idea of EDDM is to compute the "distance" between two consecutive forecast faults, in which the so-called "distance" is defined as the number of the data points that are classified correctly between two consecutive forecast faults. In ideal, if more precise results are obtained during the learning process, the distance between two consecutive forecast faults is longer. Conversely, if the distance between faults becomes shorter, it implies that a concept drift occurs.

The EDDM judges occurrence of a concept drift through whether the distance between two consecutive forecast faults has a statistically remarkable decrease. The details of EDDM are given as follows. First, let d_i be the distance between the (i – 1)th and the ith faults since the last concept drift. Then, let d_i', $d_{\max}'$, and s_i' be the average, maximum, and standard deviation of all these distances until the ith faults, respectively, since the last concept drift. And, let $s_{\max}' = \{s_1', s_2', ..., s_i'\}$. Given an α value, if the input data point is the ith forecast fault since the last concept drift, then a warning level is detected if the following inequality holds:

$$(d_i' + 2 \cdot s_i') / (d_{\max}' + 2 \cdot s_{\max}') < \alpha$$

Given a β value, a drift level is detected if the following inequality holds:

$$(d_i' + 2 \cdot s_i') / (d_{\max}' + 2 \cdot s_{\max}') < \beta$$

If the MDDD enters a warning level, the input testing data points after the warning level are stored. If the MDDD enters a drift level, occurrence of a concept drift is determined. Note that warning and drift levels represent the levels of significance of increase of the forecast fault frequency of the model at a certain time point. However, when d_i' exceeds the warning level, and d_{i+1}' is less than the warning level, these data points stored for later use are deleted, and it is regarded as

a false alarm for the warning level. That is, the warning level can not only be used for storing the data points for later use, but also be used for judge false alarms. But, the drift level is used only for judge occurrence of a concept drift.

C. Stage 3: Adaptation

When a concept drift is detected at Stage 2, Stage 3 conducts online bagging with $\lambda = 1$ (resp., $\lambda = 0.001$) on the data points collected from the warning level to the drift level to generate a major dataset called new-low-diversity (resp., new-high-diversity). Next, old-high-diversity and new-low-diversity major datasets are combined to train j hypotheses to serve as the ensemble at Stage 1 of the MDDD, and the new-high-diversity major dataset replaces the old-high-diversity major dataset. The reason why to do so is explained as follows. After a concept drift, the low-diversity ensemble trained by old-low-diversity major dataset will lose its accuracy rapidly, and need to be retrained by other major datasets. The new-low-diversity major dataset can adapt to occurrence of a concept drift rapidly. It is intuitive to retrain ensemble with new-low-diversity major dataset but it will discard all the information value of the old data, which might be useful to deal with the new concept. From experiences of previous works, the old-high-diversity major dataset with high diversity would perform better than the old-low-diversity major dataset after concept drifts. In addition, the old-high-diversity major dataset still reserves the value of old data. Hence, the two major datasets are adopted to retrain low-diversity ensemble model after a concept drift.

Finally, when a low-diversity ensemble model is trained, the new-high-diversity major dataset replaces the old-high-diversity dataset to reserve the information of this concept, and the MDDD goes back to Stage 1 for later testing.

D. MapReduce-based MDDD

Because the MDDD involves a large number of base classifiers of different types, the MDDD runs not efficiently on a PC with only one processor, which cannot meet the requirement of real CBM applications. From the literature, the MapReduce framework has been developed to increase the efficiency of ensemble learning. For instance, Hegazy et al. [13] proposed four ensemble learning models based on the MapReduce framework. Therefore, this work proposes the implementation of the MDDD based on the MapReduce framework (Fig. 2) to increase computational efficiency. The framework consists of three phases: data partition, establishment of base classifiers by Map functions, and ensemble by Reduce functions.

Phase 1 applies online bagging on D_T to generate j datasets D_1, D_2, …, D_j. For each of the j datasets, Phase 2 adopts r Mappers to serve as r training classifiers to generate r immediate results. At Phase 3, because Stage 1 of the MDDD includes outer and inner layers, we adopt two layers of Reduce functions to aggregate the results generated by all Mappers. The first layer of Reducers adopts weighted majority voting to aggregate all the r immediate results of each hypothesis. Then, the second layer of Reducer adopts unweighted majority voting to aggregate all the results into a single final result *fp*.

Because training and testing the MDDD takes the most computational time, the proposed MapReduce framework only considers the design of training and testing MDDD, and the other parts of the MDDD (e.g., establishing high-diversity and low-diversity major datasets, and detecting concept drifts) are achieved by the Master node.

When initializing the MDDD, the Master node adopts online bagging on D_T to generate old-high-diversity and old-low-diversity major datasets. Then, the MDDD adopts the old-low-diversity dataset and the Map function to establish $j \times r$ base classifiers. When a testing data point enters the Master node, we enter Stage 1 of the MDDD. The Master node passes the testing data point to each Reducer. Then, each Reducer adopts Map functions (base classifiers) to forecast the training data point, and the results are called immediate results. Then, the weighted majority voting is adopted on these immediate results to obtain final result *fp*. Then, the MDDD enters Stage 2. The Master node then adopts the EDDM to check whether the final result *fp* causes a concept drift. If the occurrence of a concept drift is confirmed, we enter Stage 3 of the MDDD. Then, online bagging is conducted on the data points stored from the warning level to the drift level to establish new-high-diversity and new-low-diversity major datasets, and combines new-low-diversity and old-high-diversity major datasets. Then, the MDDD adopts the combined major dataset and the Map function to train j classifiers to serve the ensemble for later use. Then, the MDDD goes back to Stage 1 of the MDDD.

IV. Experimental Implementation and Results

This section first introduces the experimental data, then shows the experimental analysis of the proposed MDDD, and then compares the experimental results using multiple classifier types and only one classifier type.

A. Experimental data

The experimental dataset is a benchmarking dataset used by the streaming ensemble algorithm called SEA [15], including 60,000 data points, each of which has three attributes and a real class label; the value of each attribute is a real number between 0 and 10 following a uniform distribution; two of the three attributes are relevant with the real class label, but the other attribute is irrelevant. The class label is binary. If the sum of relevant attributes is greater than a given threshold, the real label is 1; otherwise, 0. The simulation processes each data point of the dataset sequentially, to simulate the feature of data stream. For each 15,000 data points, we suppose a concept drift in which the threshold changes from one value to another value, and hence the dataset has four thresholds and three concept drifts. In the experimental setting, the thresholds are 8.0, 9.0, 7.5, and 9.5 in the ordering of time, and each 15,000 data points have the same concept. The patterns of concept drifts considered in the experiment are abrupt drifts. In addition, we assume that 10% of the data in the same concept has noise. That is, the class labels of the noise data points are opposite to the real class labels in the SEA dataset, to increase

the difficulty in forecasting the classification of this data set. As shown in Fig. 3, the SEA dataset is divided into two parts: the first 1000 data points serve as the training data, and the remaining 59,000 data points serve as the testing data. In the experiment, the MDDD reads each data point in the training dataset sequentially, forecasts the class label of the data point, and the real class labels of the data points that have been read have been known.

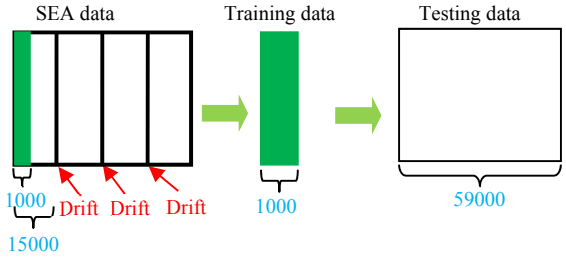

Fig 3. Illustration of partitioning experimental data.

B. Experimental analysis of the MDDD

This section analyzes whether the MDDD is successful in detecting and adapting to concept drifts, and the performance of the experimental results. The plots of accuracy versus the number of data points that has been tested are shown in Fig. 4, in which the positions of three concept drifts are marked by green dash lines (i.e., 14000, 29000, 44000); the blue solid-line curve represents the overall forecast accuracy for the data points that have been processed so far; and the red dotted-line curve represents the accuracy for the data points processed after a concept drift is detected. Note that the first concept starts from -1000 because the first 1000 data points (denoted from -1000 to 0) are reserved as the training data and thus have no accuracies. From Fig. 4, the MDDD is successful to detect the right occurrence times of three concept drifts. In addition, the MDDD can adapt to the concept drifts and raise the overall accuracy to 87.98%. In this experiment, the inner layer of MDDD consists of three types of base classifiers (i.e., LDA, NB, and CART), and the outer layer consists of 13 hypothesis, making 39 base classifiers in total.

Fig. 4. Experimental results of analyzing the accuracy of the MDDD.

C. Comparing the results between using multiple classifier types and only one classifier type

Fig. 5 shows the experimental comparison of accuracies of the proposed MDDD (i.e., with three classifier types: LDA, NB, and CART) and three MDDDs with only one single classifier type (i.e., LDA, NB, and CART, respectively). The three MDDDs with only LDAs, NBs, and CARTs are denoted by DDD-LDA, DDD-NB, and DDD-CART, respectively. From Fig. 5, the accuracy of the proposed MDDD performs better than that of the other three models.

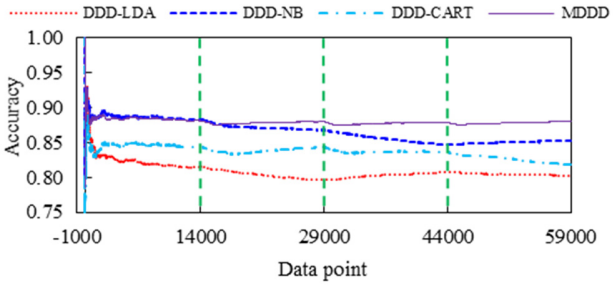

Fig. 5. Comparison of the MDDD with other classifiers.

The accuracies of the four models for the testing data points divided by three concept drifts are shown in Table 1. From Table 1, the proposed MDDD performs better than the other three MDDDs with only one classifier type in all parts except it losses DDD-NB a bit for the first part of the testing data points. The other three MDDDs with only one classifier type have no consistent conclusion on performance for different parts of the testing data points. In addition, from these experimental results, the advantage of the weighted majority voting in the MDDD can be observed. The weight (reflecting the precision) of a classifier in the ensemble is always updated to be applied, so that the overall precision increases.

TABLE. 1. ACCURACY OF FOUR MODELS FOR FOUR PARTS OF DATA

Model \ Data points	1~ 14000	14001~ 29000	29001~ 44000	44001~ 59000
DDD-LDA	0.816	0.778	0.830	0.786
DDD-NB	0.883	0.853	0.808	0.867
DDD-CART	0.843	0.843	0.821	0.766
MDDD	0.881	0.878	0.876	0.883

V. CONCLUSION

This work has proposed an MDDD ensemble learning method for CBM with concept drifts. The MDDD improves the DDD with multiple classifier types and diversity of classifiers and training data, to avoid the problem of variance and overfitting and to efficiently detect and adapt to concept drifts. To increase the computational efficiency, the implementation of the MDDD based on the MapReduce framework has been proposed. Simulation results show that the MDDD can successfully detect the right occurrence time of concept drifts and adapt to them.

REFERENCES

[1] K. Wang, Y. Wang, Y. Sun, S. Guo, J. Wu, "Green industrial Internet of things architecture: An energy-efficient perspective," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 48-54, 2016.

[2] C.-C. Lin, D.-J. Deng, J.-R. Kang, S.-C. Chang, and C.-H. Chueh, "Forecasting rare faults of critical components in LED epitaxy plants using a hybrid grey forecasting and harmony search approach," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2228-2235, 2016.

[3] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619-633, 2012.

[4] K. Wang, H. Lu, L. Shu, J.J.P.C. Rodrigues, "A context-aware system architecture for leak point detection in the large-scale petrochemical industry," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 62-69, 2014.

[5] K. Wang, L. Zhuo, Y. Shao, D. Yue, K. F. Tsang, "Toward distributed data processing on intelligent leak-points prediction in petrochemical industries," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2091-2102, 2016.

[6] H. Ke, P. Li, S. Guo, and M. Guo, "On traffic-aware partition and aggregation in MapReduce for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 818-828, 2016.

[7] I. Palit and C. K. Reddy, "Scalable and parallel boosting with MapReduce," *IEEE Transactions on Knowledge and Data Engineering* vol. 24, no. 10, pp. 1904-1916, 2012.

[8] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, Article ID: 44, 2014.

[9] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. of the 13th International Conference on Machine Learning (ICML 1996)*, pp. 148-156, 1996.

[10] L. L. Minku, A. P. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drift," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 730-742, 2010.

[11] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517-1531, 2011.

[12] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Proc. of 4th International Workshop on Knowledge Discovery from Data Streams*, vol. 6, pp. 77-86, 2006.

[13] O. Hegazy, S. Safwat, and M. El Bakry, "A MapReduce fuzzy techniques of big data classification," in *Proc. of SAI Computing Conference (SAI)*, IEEE Press, pp. 118-128, 2016.

[14] N. C. Oza, "Online bagging and boosting," in *Proc. of 2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2340-2345, 2005.

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