Concept Drift

Marcus A. Maloof
Georgetown University, USA

INTRODUCTION

Traditional approaches to data mining are based on an assumption that the process that generated or is generating a data stream is static. Although this assumption holds for many applications, it does not hold for many others. Consider systems that build models for identifying important e-mail. Through interaction with and feedback from a user, such a system might determine that particular e-mail addresses and certain words of the subject are useful for predicting the importance of e-mail. However, when the user or the persons sending e-mail start other projects or take on additional responsibilities, what constitutes important e-mail will change. That is, the concept of important e-mail will change or drift. Such a system must be able to adapt its model or concept description in response to this change. Coping with or tracking concept drift is important for other applications, such as market-basket analysis, intrusion detection, and intelligent user interfaces, to name a few.

BACKGROUND

Concept drift may occur suddenly, what some call *revolutionary drift.* Or it may occur gradually, what some call *evolutionary drift* (Klenner & Hahn, 1994). Drift may occur at different time scales and at varying rates. Concepts may change and then reoccur, perhaps with contextual cues (Widmer, 1997). For example, the concept of warm is different in the summer than in the winter. A contextual cue or variable is perhaps the season or the daily mean temperature. It helps identify the appropriate model to use for determining if, for example, it is a warm day. Coping with concept drift requires an online approach, meaning that the system must mine a stream of data. If drift occurs slowly enough, distinguishing between real and virtual concept drift may be difficult (Klinkenberg & Joachims, 2000). The former occurs when concepts indeed change over time; the latter occurs when performance drops during the normal online process of building and refining a model. Such drops could be due to differences in the data in different parts of the stream. The richness of this problem has led to an equally rich collection of approaches.

Research on the problem of concept drift has been empirical and theoretical (see Kuh, Petsche, & Rivest, 1991; Mesterharm, 2003); however, the focus of this article is on empirical approaches. In this regard, researchers have evaluated such approaches by using synthetic data sets (for examples, see Maloof & Michalski, 2004; Schlimmer & Grainger, 1986; Street & Kim, 2001; Widmer & Kubat, 1996) and real data sets (for examples, see Black & Hickey, 2002; Blum, 1997; Lane & Brodley, 1998), both small (Maloof & Michalski, 2004; Schlimmer & Grainger, 1986; Widmer & Kubat, 1996) and large (Hulten, Spencer, & Domingos, 2001; Kolter & Maloof, 2003; Street & Kim, 2001; Wang, Fan, Yu, & Han, 2003).

Systems for coping with concept drift fall into three broad categories: incremental, partial memory, and ensemble. Incremental approaches use new instances to modify existing models. Partial-memory approaches maintain a subset of previously encountered instances, previously built and refined models, or both. When new instances arrive, such systems use the new instances and their store of past instances and models to build new models or refine existing ones. Finally, ensemble methods build and maintain multiple models to cope with concept drift. Naturally, systems designed for concept drift do not always fall neatly into only one of these categories. For example, some partial-memory approaches are also incremental (Maloof & Michalski, 2004; Widmer & Kubat, 1996).

A *model* or *concept description* is a generalized representation of instances or training data. Such representations are important for prediction and for better understanding the data set from which they were built. Researchers have used a variety of representations for drifting concepts, including the instances themselves, probabilities, linear equations, decision trees, and decision rules. Methods for inducing and building such representations include instance-based learning, naïve Bayes, support vector machines, C4.5, and AQ15, respectively. See Hand, Mannila, and Smyth (2001) for additional information.

MAIN THRUST

Researchers have proposed a variety of approaches for learning concepts that drift. However, evaluating such

approaches is a critical issue. In the next two sections, I survey approaches for learning concepts that change over time and discuss issues of evaluating such approaches.

Survey of Approaches for Concept Drift

STAGGER (Schlimmer & Grainger, 1986) was the first system for coping with concept drift. Its model consists of nodes, corresponding to features and class labels, linked together with probabilistic arcs, representing the strength of association between features and class labels. As STAGGER processes new instances, it increases or decreases probabilities, and it may add nodes and arcs. To classify an unknown instance, STAGGER predicts the most probable class.

Partial-memory approaches maintain a store of partially built models, a portion of the previously encountered instances, or both. Such approaches vary in how they use such information for adjusting current models. The FLORA Systems (Widmer & Kubat, 1996) maintain a sequence of examples over a dynamically adjusted window of time. The Window Adjustment Heuristic (WAH) adjusts the size of this window in response to performance changes. Generally, if performance is decreasing or poor, then the heuristic reduces the window's size; if it is increasing or acceptable, then it increases the size. These systems also maintain a store of rules, including ones that are overly general, although these are not used for prediction. As the systems process instances, they create new rules or refine existing ones. To classify an instance, the FLORA systems select the rule that best matches and return its class label.

The AQ-PM Systems maintain a set of examples over a window of time, but the systems select examples from the boundaries of rules, so they can retain examples that do not reoccur in the data stream. AQ-PM (Maloof & Michalski, 2000) builds new rules when new examples arrive, whereas AQ11-PM (Maloof & Michalski, 2004) refines existing rules. These systems maintain examples over a static window of time, but AQ11-PM+WAH (Maloof, 2003) incorporates Widmer and Kubat's (1996) WAH for dynamically sizing this window. Because these systems use rules, when classifying an instance, they return as their prediction the class label of the best matching rule.

The Concept-adapting Very Fast Decision Tree system (Hulten et al., 2001), or CVFDT, progressively grows a decision tree downward from the leaf nodes. It maintains frequency counts for attribute values by class and extends the tree when a statistical test indicates that a change has occurred. CVFDT also maintains at each node a list of alternate subtrees, which it swaps with the current subtree when it detects drift. To classify an unknown instance, the method uses the instance's values to traverse the current tree from the root to a leaf node, returning as the prediction the associated label.

The Concept Drift 3 system (Black & Hickey, 1999), or CD3, uses batches of instances annotated with a time stamp of either *current* or *new* to build a decision tree. When drift occurs, time becomes more relevant for prediction, so the time-stamp attribute will appear higher in the decision tree. After pruning, CD3 converts the tree to rules by enumerating all paths containing a new time stamp and then removing conditions involving the time stamp. CD3 predicts the class of the best matching rule.

Ensemble methods maintain a set of models and use a voting procedure to yield a global prediction. Blum's (1997) implementation of Weighted-Majority (Littlestone & Warmuth, 1994) uses as models histories of labels associated with pairs of features. If a model's features are present in an instance, then it predicts the most frequent label present in its history. The method initializes each model with a weight of 1 and reduces a model's weight if it predicts incorrectly. It predicts based on a weighted vote of the predictions of the models.

The Streaming Ensemble Algorithm (Street & Kim, 2001) maintains a fixed-size collection of models, each built from a fixed number of instances. When a new batch of instances arrives, SEA builds a new model. If space exists in the collection, then it adds the new model. Otherwise, it replaces the worst performing model with the new model, if one exists. SEA predicts the majority vote of the predictions of the models in the collection.

The Accuracy-Weighted Ensemble (Wang et al., 2003) also maintains a fixed-size collection of models, each built from a batch of instances. However, this method weights each classifier in the collection based on its performance on the most recent batch. When adding a new weighted model, if there is no space in the collection, then the method stores only the top weighted models. The method predicts based on a weighted-majority vote of the predictions of the models in the collection.

Dynamic Weighted Majority (Kolter & Maloof, 2003), or DWM, maintains a collection of weighted models but dynamically adds and removes models with changes in performance. Instead of building a single model with each batch, DWM uses new instances to refine all the models in the collection. Each time a model predicts incorrectly, DWM reduces its weight, and DWM removes a model from the collection if its weight falls below a threshold. Like the previous method, DWM predicts based on a weighted-majority vote of the predictions of the models, but if the global prediction