StreamMiner: A Classifier Ensemble-based Engine to Mine Concept-drifting Data Streams

Wei Fan

IBM T.J.Watson Research
19 Skyline Drive
Hawthorne, NY 10532, USA
weifan@us.ibm.com

Abstract

We demonstrate StreamMiner, a random decision-tree ensemble based engine to mine data streams. A fundamental challenge in data stream mining applications (e.g., credit card transaction authorization, security buy-sell transaction, and phone call records, etc) is concept-drift or the discrepancy between the previously learned model and the true model in the new data. The basic problem is the ability to judiciously select data and adapt the old model to accurately match the changed concept of the data stream. StreamMiner uses several techniques to support mining over data streams with possible concept-drifts. We demonstrate the following two key functionalities of StreamMiner:

1. Detecting possible concept-drift on the fly when the trained streaming model is used to classify incoming data streams *without* knowing the ground truth.
2. Systematic data selection of old data and new data chunks to compute the optimal model that best fits on the changing data streams.

1 Introduction

One of the recent challenges facing traditional data mining methods is to handle real-time production systems that produce large amount of data continuously at unprecedented rate and with evolving patterns. Traditionally, due to limitation of storage and practitioner's ability to mine huge amount of data, it is a common practice to mine a subset of data at preset frequency. However, these solutions have been shown to be ineffective due to possibly over-simplified model as a result of sub-sampling as well as dynamically unpredictable evolving pattern of the production data. Knowledge discovery on data streams has become a research topic of growing interest. Much work has been done on modeling [Babcock et al., 2002], querying [Babu and Widom, 2001, Gao and Wang, 2002, Greenwald and Khanna, 2001], classification [Hulten et al., 2001, Street and Kim, 2001, Wang et al., 2003, Fan et al., 2004, Fan, 2004b], regression analysis [Chen et al., 2002] and clustering [Guha et al., 2000]. The fundamental problem is the following: given an infinite amount of continuous measurements, how do we model them in order to capture possibly time-evolving trends and patterns in the stream, compute the optimal model and make time critical decisions?

2 The Motivation of StreamMiner

The fundamental problem in learning drifting concepts is how to identify in a timely manner those data in the training set that are no longer consistent with the current concepts. These data must be discarded. A straightforward solution, which is used in many current approaches, discards data indiscriminately after they become old, that is, after a fixed period of time T has passed since their arrival. Although this solution is conceptually simple, it tends to complicate the logic of the learning algorithm. More importantly, it creates the following dilemma which makes it vulnerable to unpredictable conceptual changes in the data: if T is large, the training set is likely to contain outdated concepts, which reduces classification accuracy; if T is small, the training set may not have enough data, and as a result, the learned model will likely carry a large

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

variance due to overfitting.

We use a simple example to illustrate the problem. Assume a stream of 2-dimensional data is partitioned into sequential chunks based on their arrival time. Let S_i be the data that came in between time t_i and t_{i+1}. Figure 1 shows the distribution of the data and the optimum decision boundary during each time interval.

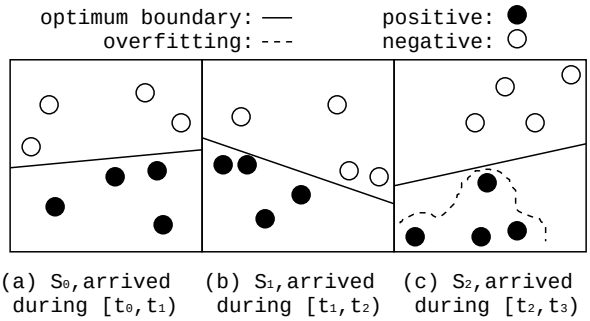

Figure 1: data distributions and optimum boundaries

The problem is: after the arrival of S_2 at time t_3, what part of the training data should still remain influential in the current model so that the data arriving after t_3 can be most accurately classified?

On one hand, in order to reduce the influence of old data that *may* represent a different concept, we shall use nothing but the most recent data in the stream as the training set. For instance, use the training set consisting of S_2 only (i.e., $T = t_3 - t_2$, data S_1, S_0 are discarded). However, as shown in Figure 1(c), the learned model may carry a significant variance since S_2's insufficient amount of data are very likely to be overfitted.

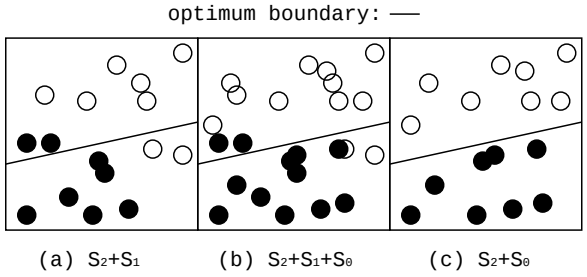

Figure 2: Which training dataset to use?

The inclusion of more historical data in training, on the other hand, may also reduce classification accuracy. In Figure 2(a), where $\mathsf{S}_2 \cup \mathsf{S}_1$ (i.e., $T = t_3 - t_1$) is used as the training set, we can see that the discrepancy between the underlying concepts of S_1 and S_2 becomes the cause of the problem. Using a training set consisting of $\mathsf{S}_2 \cup \mathsf{S}_1 \cup \mathsf{S}_0$ (i.e., $T = t_3 - t_0$) will not solve the problem either. Thus, there may not exists an optimum T to avoid problems arising from overfitting and conflicting concepts.

We should not discard data that may still provide useful information to classify the current test examples. Figure 2(c) shows that the combination of S_2 and S_0 creates a classifier with less overfitting or conflicting-concept concerns. The reason is that S_2 and S_0 have similar class distribution. Thus, instead of discarding data using the criteria based solely on their arrival time, we shall make decisions based on their class distribution. Historical data whose class distributions are similar to that of current data can reduce the variance of the current model and increase classification accuracy. However, it is a non-trivial task to select training examples based on their class distribution.

3 The StreamMiner Solution

There are a large number of possibilities that can happen when mining data streams. Before we go into the details of the main mining engine, we enumerate all situations that we can think of and discuss the best choice in each case and how to find the optimal model. The two main themes of our comparison is on possible data insufficiency and concept drift. We start from simple cases.

- **New data is sufficient by itself *and* there is no concept drift.** The optimal model should be the one trained from the new data itself since new data is sufficient. The older model may also be an optimal model if it is trained from sufficient data. However, the tricky issue is that we do not know and will usually never know if the data is indeed sufficient and the concept indeed remains the same. However, it doesn't hurt to train a new model from the new data, a new model from combined new data and old data, and compare with the original older model to choose the more accurate one if the learning cost is affordable.

- **New data is sufficient by itself *and* there is concept drift.** The optimal model should be the one trained from the new data itself. Similar to the previous situation, we do not know and will never know if the data is indeed sufficient and the concept indeed remains the same. Ideally, we should compare a few sensible choices if the training cost is affordable.

- **New data is insufficient by itself *and* there is no concept drift.**. If the previous data is sufficient, the optimal model should be the existing model. Otherwise, we should train a new model from new data plus existing data and choose the one with higher accuracy.

- **New data is insufficient by itself *and* there is concept drift.** Obviously, training a new model from new data only doesn't return the optimal model. However, choosing old data unse-

lectively, as shown previously, will only be misleading. The correct approach is to choose only those examples from previous data chunks that have consistent concept with the new data chunk and combine those examples with the new data

3.1 Computing optimal models

We notice that the optimal model is completely different under different situations. The choice for optimal model completely depends on if the data is indeed sufficient and if there is indeed concept drift. The ideal solution would be to compare a few plausible optimal models statistically, and choose the one with the highest accuracy. To clarify some notation conventions, $FN(\mathbf{x})$ denotes a new model trained from recent data. $FO(\mathbf{x})$ denotes an optimal model finally chosen after some statistical significance tests. i is the sequence number of each sequentially received data chunk.

1. Train a model $FN_i(\mathbf{x})$ from the new data chunk S_i only.

2. Assume that D_{i-1} is the dataset that trained the most recent "optimal" model $FO_{i-1}(\mathbf{x})$. It is important to point out that D_{i-1} may not be the most recent data chunk S_{i-1}. D_{i-1} is collected iteratively throughout the streaming data mining process. The exact way how D_{i-1} is collected will be clear next. We select these examples from D_{i-1} that both the trained new model $FN_i(\mathbf{x})$ and the recent optimal model $FO_{i-1}(\mathbf{x})$ make the correct prediction. We denote these chosen examples as s_{i-1}. In other words, $s_{i-1} = \{\forall(\mathbf{x}, y) \in D_{i-1}, \text{such that}, (FN_i(\mathbf{x}) = y) \wedge (FO_{i-1}(\mathbf{x}) = y)\}$.

3. Train a model $FN_i^+(\mathbf{x})$ from the new data *plus* the selected data in the last step or $S_i \cup s_{i-1}$.

4. Update the most recent model FO_{i-1} with S_i and call this model $FO_{i-1}^+(\mathbf{x})$. To update a model, we keep the "structure" of the model and update its internal statistics. Using decision tree as an example, every example in S_i is "classified" or sorted to each leaf node. The statistics, i.e., the number of examples belonging to each class label, are updated. Obviously, the training set for $FO_{i-1}^+(\mathbf{x})$ is $D_i \cup S_i$.

5. Compare the accuracy of all four models ($FN_i(\mathbf{x})$, $FO_{i-1}(\mathbf{x})$, $FN_i^+(\mathbf{x})$), and $FO_{i-1}^+(\mathbf{x})$) using "cross-validation" and choose the one that is the most accurate and we name it $FO_i(\mathbf{x})$.

6. D_i is the training set that computes $FO_i(\mathbf{x})$. It is one of $S_i, D_{i-1}, S_i \cup s_{i-1}$, and $S_i \cup D_{i-1}$.

3.2 Main Engine

The main engine of StreamMiner trains a number of random and uncorrelated decision trees. Details of the main engine can be found in [Fan et al., 2003, Fan, 2004a, Fan, 2004b]. Each decision tree is constructed by randomly selecting available features. The structure of the tree is uncorrelated. Their only correlation is on the training data itself. To classify an example, raw posterior probability is required. If there are n_c examples out of n in the leaf node with class label c, the probability that $\mathbf{x}$ is an example of class label c is $P(c|\mathbf{x}) = \frac{n_c}{n}$. Each tree computes a posterior probability for an example and the probability outputs from multiple trees are averaged as the final posterior probability of the ensemble. To make a decision, application specific loss function is required. For a binary problem under 0-1 loss, if $P(y|\mathbf{x}) > 0.5$, the best prediction is y.

Cross-validation is implemented by using the model itself. Assuming that n is the size of the training set, n-fold cross validation leaves one example $\mathbf{x}$ out and uses the remaining $n-1$ examples to train a model and classify on the left-out example $\mathbf{x}$. When we compute the probability for the excluded $\mathbf{x}$ under n-fold cross validation using the original decision tree ensemble, we need to compensate this difference. Assuming that we have two class labels, either fraud or non-fraud, to compute the probability of the excluded $\mathbf{x}$ being fraudulent is simply

$$\begin{cases} \frac{n_{fraud}-1}{n_{fraud}-1+n_{normal}} & \text{if } \mathbf{x} \text{ is indeed a fraud} \\ \frac{n_{fraud}}{n_{fraud}+n_{normal}-1} & \text{if } \mathbf{x} \text{ is a normal transaction} \end{cases}$$

4 About this Demo

Streaming Data Generator

We create synthetic data with drifting concepts based on a moving hyperplane that is commonly used to simulate concept-drifting data streams. A hyperplane in d-dimensional space is denoted by equation: $\sum_{i=1}^{d} a_i x_i = a_0$. We label examples satisfying $\sum_{i=1}^{d} a_i x_i \geq a_0$ as positive, and examples satisfying $\sum_{i=1}^{d} a_i x_i < a_0$ as negative. Hyperplanes have been used to simulate time-changing concepts because the orientation and the position of the hyperplane can be changed in a smooth manner by changing the magnitude of the weights [Hulten et al., 2001]. We generate random examples uniformly distributed in multi dimensional space $[0, 1]^d$. Weights a_i $(1 \leq i \leq d)$ are initialized randomly in the range of $[0, 1]$. We choose the value of a_0 so that the hyperplane cuts the multi-dimensional space in two parts of the same volume, that is, $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$. Thus, roughly half of the examples are positive, and the other half negative. Noise is introduced by randomly switching the labels of $p\%$ of the examples. In our experiments, the noise level $p\%$ is set to 5%.

We provide a few parameters that the attendees of our demo can choose to simulate different degrees of concept-drift. Parameter k specifies the total number

of dimensions whose weights are changing. Parameter $t \in \mathcal{R}$ specifies the magnitude of the change (every N examples) for weights $a_1, \cdots, a_k$, and $s_i \in \{-1, 1\}$ specifies the direction of change for each weight a_i, $1 \leq i \leq k$. Weights change continuously, i.e., a_i is adjusted by $s_i \cdot t/N$ after each example is generated. Furthermore, there is a possibility of 10% that the change would reverse direction after every N examples are generated, that is, s_i is replaced by $-s_i$ with probability 10%. Also, each time the weights are updated, we recompute $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$ so that the class distribution is not disturbed.

Concept Change Illustration

Conceptual change is best illustrated through the change of error rate of models. For every historically trained model, we show its changing error rate on the evolving data stream. Based on the attendee's parameter selection, the trend we will show is that models trained from recent data and systematically selected old data will have a generally lower error rate than the older models.

5 Demonstration Scenario

In our demo, the attendees have the freedom to choose different parameters to simulate a data stream with changing concept and the amount of new data collected until a new model need to be learned. After the attendee chooses these parameters, the stream data generator will produce incoming data streams. StreamMiner collects the data continuously and starts to compute or update an existing model when the number of new examples are above a threshold the attendee chooses. This process can either run continuously continuously or in batch mode, as chosen by the demo attendee.

Acknowledgement

The original synthetic data generator was written by my colleague, Dr. Haixun Wang, as described in a previous paper [Wang et al., 2003]. The original source was modified to generate continuous data streams with drifting concepts.

References

[Babcock et al., 2002] Babcock, B., Babu, S., Datar, M., Motawani, R., and Widom, J. (2002). Models and issues in data stream systems. In *ACM Symposium on Principles of Database Systems (PODS)*.

[Babu and Widom, 2001] Babu, S. and Widom, J. (2001). Continuous queries over data streams. *SIGMOD Record*, 30:109–120.

[Chen et al., 2002] Chen, Y., Dong, G., Han, J., Wah, B. W., and Wang, J. (2002). Multi-dimensional regression analysis of time-series data streams. In *Proc. of Very Large Database (VLDB)*, Hongkong, China.

[Fan, 2004b] Fan, W. (August 2004b). Systematic data selection to mine concept-drifting data streams. In *Proceedings of 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2004)*, Seattle, Washington, USA.

[Fan, 2004a] Fan, W. (July 2004a). On the optimality of probabililty estimation by random decision trees. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'2004)*, San Jose, California, USA.

[Fan et al., 2004] Fan, W., an Huang, Y., Wang, H., and Yu, P. S. (April 2004). Active mining of data streams. In *Proceedings of 2004 SIAM International Conference on Data Mining*, pages 457–461.

[Fan et al., 2003] Fan, W., Wang, H., Yu, P. S., and Ma, S. (2003). Is random model better? on its accuracy and efficiency. In *Proceedings of Third IEEE International Conference on Data Mining (ICDM'2003)*.

[Gao and Wang, 2002] Gao, L. and Wang, X. (2002). Continually evaluating similarity-based pattern queries on a streaming time series. In *Int'l Conf. Management of Data (SIGMOD)*, Madison, Wisconsin.

[Greenwald and Khanna, 2001] Greenwald, M. and Khanna, S. (2001). Space-efficient online computation of quantile summaries. In *Int'l Conf. Management of Data (SIGMOD)*, pages 58–66, Santa Barbara, CA.

[Guha et al., 2000] Guha, S., Milshra, N., Motwani, R., and O'Callaghan, L. (2000). Clustering data streams. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 359–366.

[Hulten et al., 2001] Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 97–106, San Francisco, CA. ACM Press.

[Street and Kim, 2001] Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)*.

[Wang et al., 2003] Wang, H., Fan, W., Yu, P., and Han, J. (2003). Mining concept-drifting data streams with ensemble classifiers. In *Proceedings of ACM SIGKDD International Conference on knowledge discovery and data mining (SIGKDD2003)*, pages 226–235.