The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

A Framework for Multistream Regression with Direct Density Ratio Estimation

Ahsanul Haque, Hemeng Tao, Swarup Chandra, Jie Liu, Latifur Khan
Department of Computer Science
University of Texas at Dallas, Richardson TX
{ahsanul.haque, hxt160430, swarup.chandra, jxl164830, lkhan}@utdallas.edu

Abstract

Regression over a stream of data is challenging due to unbounded data size and non-stationary distribution over time. Typically, a traditional supervised regression model over a data stream is trained on data instances occurring within a short time period by assuming a stationary distribution. This model is later used to predict value of response-variable in future instances. Over time, the model may degrade in performance due to changes in data distribution among incoming data instances. Updating the model for change adaptation requires true value for every recent data instances, which is scarce in practice. To overcome this issue, recent studies have employed techniques that sample fewer instances to be used for model re-training. Yet, this may introduce sampling bias that adversely affects the model performance. In this paper, we study the regression problem over data streams in a novel setting. We consider two independent, yet related, non-stationary data streams, which are referred to as the source and the target stream. The target stream continuously generates data instances whose value of response variable is unknown. The source stream, however, continuously generates data instances along with corresponding value for the response-variable, and has a biased data distribution with respect to the target stream. We refer to the problem of using a model trained on the biased source stream to predict the response-variable's value in data instances occurring on the target stream as *Multistream Regression*. In this paper, we describe a framework for multistream regression that simultaneously overcomes distribution bias and detects change in data distribution represented by the two streams over time using a Gaussian kernel model. We analyze the theoretical properties of the proposed approach and empirically evaluate it on both real-world and synthetic data sets. Importantly, our results indicate superior performance by the framework compared to other baseline regression methods.

Introduction

Distribution of data generated continuously from a non-stationary domain may change over time due to various internal or external factors. Examples of such non-stationary domains include financial transactions, telephone calls, readings from sensor networks, etc. A predictor over streams of data should adapt to changes in distribution, for avoiding performance degradation, by appropriately adjusting its target function. Moreover, traditional supervised predictors may require storage of all historical data. This is not practical in a data stream due to limited memory.

Studies in data stream classification (Ikonomovska et al. 2009; Masud et al. 2011) have proposed multiple techniques that adapt a predictor's target function to changes in data distribution over time. This change is referred to as *Concept Drift* (Gama et al. 2014). In general, a sliding window-based technique is used to detect a change over time. Initially, a predictor is trained in a supervised manner, using sequential data instances observed within a window of a finite size. This predictor is used to evaluate the response value of each new data instance occurring in subsequent windows along the stream. When a change in data distribution is detected, the target function is re-evaluated using a training set formed by data instances in the current window. This technique can be applied to the regression problem as well. However, such a technique assumes availability of true target values for all recent data instances. Collecting true values often require human effort, typically resulting in a costly and time consuming process. Therefore, these techniques may not be suitable in real-world scenarios under scarcity of true target values, where data instances occur at a high velocity.

Instead of obtaining the true target value for each data instance soon after prediction, active learning based mechanisms selectively sample from recent data instances for truth extraction (Haque, Khan, and Baron 2016; Fan et al. 2004). However, a sampling technique may induce bias in sampled data distribution. Such bias can be introduced due to reasons such as availability and cost of obtaining truth values. For example, consider the scenario of predicting future household electricity consumption for optimal distribution. Data from each household only include the current power consumption, and not a future estimate. Since it is an expensive process to require each household to continuously provide an estimate of future power consumption, there is a tendency to choose certain households to collect such estimates for monitoring predictor performance and adapting to distribution changes when necessary. However, such a set of households may result in under-representative samples for a good long-term prediction performance in the whole market. Therefore, there is a need to correct bias before using the data for predictor adaptation.

In this paper, we perform regression in a setting involving two types of non-stationary data streams. A *source* stream continuously generates data instances containing both independent and dependent variables, i.e., true response value for each data instance is known. However, data distribution of the source stream is biased compared to a *target* stream, which generates data instances from the same domain but whose true response values are unknown. The regression problem is to predict response values of data instances in the target stream utilizing information available in the source stream. This problem is referred to as the *Multistream Regression*. Importantly, the non-stationary nature of the two independent streams result in asynchronous concept drifts.

The main challenge in multistream regression is to effectively correct bias between the source and target streams, whilst continuously address concept drifts along the stream. Our goal is to efficiently use the available response values in the source stream for evaluating a target function associated with data instances in the target stream Naively combining the two streams into a single stream for employing existing prediction techniques may not be effective since the combined stream results in an overall data distribution, which attenuates the effect of individual drifts and adversely affects prediction performance. The main contributions of our work are as follows. (1) We perform regression in a multistream setting by utilizing the bias-corrected source stream data to train a prediction model, which is used over target stream data. (2) We propose an online regression model which utilizes an efficient technique that simultaneously performs bias correction and asynchronous concept drift detection between source and target stream data. Particularly, we utilize a Gaussian kernel model for continuously estimating density ratios, whose output is used to update our regression model. (3) We analyze the theoretical properties of our approach to show its effectiveness in addressing the challenges of multistream regression setting. (4) We evaluate our technique on both synthetic and real-world datasets, and compare the results with baseline methods.

The paper is organized as follows. We first discuss related studies and background in data stream mining and transfer learning, and then list the set of challenges in the multistream regression problem. Next, we present our framework to address these challenges, and analyze its theoretical properties. Finally, we empirically evaluate the framework on numerous data sets and conclude.

Related Work

Concept Drift Most data stream mining approaches in the recent past have focused on addressing the concept drift problem on a single stream while performing data classification or regression. Essentially, data in the stream is viewed within a finite time window wherein its distribution is stationary. Particularly with respect to regression, the target function is obtained using a training data within a window, and is used to evaluate future data instances in the stream. A feedback mechanism using prediction error (Gama et al. 2004) or confidence (Haque et al. 2016) is used to detect changes in distribution between two different time windows by detecting change points in the process. An approach by (Ikonomovska and Gama 2008) constructs regression trees from streaming data by incrementally adding data items. (Rosenthal et al. 2009) proposed an ensemble regression approach for drifting processes. (Nadungodage et al. 2014) investigated an approach called ESR (Ensemble Stream Regression), which trained an ensemble of regression models from sequential chunks of instances in a data stream and dynamically re-computed the regression function parameters. Apart from working on a single stream, these methods explicitly detect change points.

Covariate Shift In traditional machine learning, the distribution of training and test data instances are assumed to be similar (Zadrozny 2004). Therefore, these techniques cannot be directly employed when such an assumption is not valid. *Covariate shift* or Sampling bias is one such case where the training data is biased with respect to the test data set, resulting in dissimilar data distribution. Instance weighting is a bias correction technique, where a weight is associated with each training data by estimating the density ratios between the test and the training distributions. Concretely, if P_{tr} and P_{te} are the probability density functions of the training and test distributions respectively, then for each training instance x, a weight is given by $\beta(x) = \frac{P_{te}(x)}{P_{tr}(x)}$. Often, popular techniques such as Kernel Mean Matching (KMM) (Huang et al. 2007), and Kullback-Leibler Importance Estimation Procedure (KLIEP) (Sugiyama et al. 2007) are used in the literature for directly estimating density ratio (β). These, along with a recently proposed a robust regression model (Chen et al. 2016), are known to work well on small datasets. In the case of multistream setting, (Chandra et al. 2016) introduced a framework that uses KMM for bias correction, with explicit concept drift detection. This was further enhanced by using a direct density ratio estimation method in (Haque et al. 2017), inspired from the change detection technique in (Kawahara and Sugiyama 2012). However, they are specific to the classification problem. Instead, we adapt these techniques for regression over a multistream setting using the Importance Weighted Least Squared (IWLS) method.

Problem Formulation

In this section, we formalize our regression problem and discuss its challenges.

Problem A data instance is denoted by (x, y), where $x \in D^v$ is a vector of v independent features, and y is its corresponding output (or dependent response) value. We assume that the streams are generated from two independent non-stationary processes in the same domain (denoted by D). We observe them through a window, denoted by $\mathbf{W}$, of size N. In the source stream, both x and y are observed in window $\mathbf{W}_S$. On the contrary, only x can be observed for each instance in target stream's window $\mathbf{W}_T$. Our goal is to use $(x, y) \in S$ and $x \in T$ for learning a target function useful for predicting y in T.

Challenges Since data is generated continuously from non-stationary processes, it is impossible to train a regression model in the traditional manner that require storing the entire data stream on the memory. Moreover, conditional

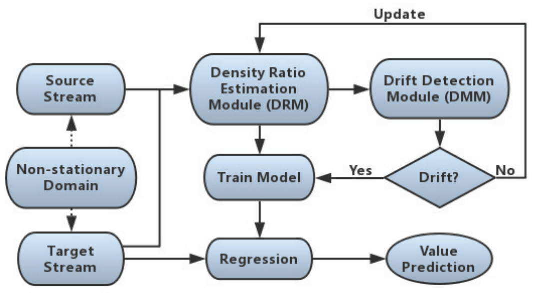


Figure 1: Multistream Regression Overview

probability distribution of data may change over time, i.e. $P^{(o)}(y|x) \neq P^{(t)}(y|x)$, where $t > o$ are observation times. This is typically known as concept drift. Moreover, in the multistream setting, such a change may occur independently over the streams. Existence of covariate shift between the source and target data complicates this dynamic scenario. By definition, the two streams have unequal covariate distribution, but equal conditional distribution, i.e., $P_S(x) \neq P_T(x)$ and $P_S(y|x) = P_T(y|x)$. Therefore, the combination of concept drift and covariate shift in the multistream regression setting is challenging.

We address these challenges by performing regression over finite-size sliding windows $\mathbf{W}_S$ and $\mathbf{W}_T$. Particularly, ordinary least square (OLS) regression uses a linear relationship between the independent and dependent variables, i.e., $y = b^T x + a$. Due to covariate shift, we use a weighted variation of least square regression, where an instance weight (denoted by β) indicates the influence of corresponding training data for determining regression parameters (Natrella 2010). The IWLS regression with n training instances is given by:

$$Q = \sum_{i=1}^{n} \beta_i [y_i - (b^T x + a)]^2 \tag{1}$$

The Proposed Approach

An overview of the proposed approach, referred to as *MultiStream Regression (MSR)*, is illustrated in Figure 1. Data instances in the source and target streams occur simultaneously. Initially, we perform bias correction by estimating the density ratio $\beta(x)$ for each incoming source data instance x in the *Density Ratio Estimation Module* (DRM). Particularly, we estimate the density ratio directly using a Gaussian kernel model. As new instances arrive, we update the model parameters online. The *Drift Detection Module* (DDM) detects a change point if there is a significant difference between the weighted training and the test distribution. Once a change point is detected, we train a new regression model using data instances in the current $\mathbf{W}_S$ and $\mathbf{W}_T$. The model is used to perform regression for data instances in T. Whilst we also learn a new β function for further density ratio estimation. We now present each module in detail.

Density Ratio Estimation Module (DRM)

We model instance weights using Gaussian kernels, where current instances in $\mathbf{W}_T$ work as the Gaussian centers, similar to (Sugiyama et al. 2007). Therefore, weight for any instance x, denoted by $\beta(x)$, can be approximated as:

$$\hat{\beta}(x) = \sum_{j=1}^{N} \alpha_j K_\sigma(x, x_T^{(j)}) \tag{2}$$

where $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^N$ are the set of parameters needed to be learned, $K_\sigma(\cdot,\cdot)$ is the Gaussian kernel, i.e., $K_\sigma(x^{(i)}, x^{(j)}) = \exp\left\{-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right\}$, and σ is the kernel width.

Algorithm 1 Density Ratio Estimation

```
Require: $\{x_S^{(i)}\}_{i=1}^N$: Source Window Instances; $\{x_T^{(j)}\}_{j=1}^N$:
    Target Window Instances; $K_\sigma(x_S, x_T)$: Gaussian Kernel Function;
Ensure: $\hat{\beta}(x_S^{(i)})$
 1: $K_{i,j} = K_\sigma(x_S^{(i)}, x_T^{(j)})$;
 2: $b_k = \frac{1}{N}\sum_{i=1}^N K_\sigma(x_S^{(i)}, x_T^{(j)})$ ;
 3: Initialize $\boldsymbol{\alpha}(> 0)$ and $\epsilon(0 < \epsilon << 1)$;
 4: repeat
 5:     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \epsilon K^{\mathrm{T}}(1/.\boldsymbol{\alpha})$;
 6:     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + (1 - b^{\mathrm{T}}\boldsymbol{\alpha})b \setminus (b^{\mathrm{T}})$;
 7:     $\boldsymbol{\alpha} \leftarrow max(0, \boldsymbol{\alpha})$;
 8:     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/(b^{\mathrm{T}}\boldsymbol{\alpha})$;
 9: until $Convergence$
10: $\hat{\beta}(x_S^{(i)}) \leftarrow \sum_{j=1}^N \alpha_j K_\sigma(x_S^{(i)}, x_T^{(j)})$
```

The target distribution is estimated by the weighted training distribution, i.e., $\hat{P}_T(x) = \hat{\beta}(x)P_S(x)$. The parameters $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^N$ are learned by minimizing the Kullback-Leibler divergence between $\hat{P}_T(x)$ and $P_T(x)$. The concave optimization version of this problem is given by:

$$\begin{aligned} &\underset{\{\alpha_j\}_{j=1}^N}{\text{maximize}} \left[\sum_{i=1}^{N} \log\left(\sum_{j=1}^{N} \alpha_j K_\sigma(x_T^{(i)}, x_T^{(j)})\right)\right] \\ &\text{subject to } \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_j K_\sigma(x_S^{(i)}, x_T^{(j)}) = 1, \\ &\text{and } \alpha_1, \alpha_2, ..., \alpha_N \geq 0. \end{aligned} \tag{3}$$

A pseudo-code for density ratio estimation is described in Algorithm 1. It will be used for initialization and re-initialization purpose when a drift is detected. We discuss the drift detection technique later in this section. The kernel width σ is determined from data samples using likelihood cross validation. The optimum solution of the parameters $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^N$ can be obtained by first performing gradient ascent (line 5) and then feasibility satisfaction, iteratively.

Updating Parameters Online

Numerous online techniques have been proposed for updating $\boldsymbol{\alpha}$. For example, (Kawahara and Sugiyama 2012) proposed a nonparametric approach to detect changes in probability distribution of a data sequence by directly estimating probability density ratios, rather than estimating the individual densities. We adapt and improve this approach for the multistream regression setting.

A new instance arriving in S only affects the constraints of the optimization problem. Whereas, a new instance in T directly affects the objective function as well. Thus, $\boldsymbol{\alpha}$ needs to be updated along with constraint satisfaction under this case. In MSR, $\boldsymbol{\alpha}$ is updated based on an online learning technique for kernel methods proposed by (Kivinen, Smola, and Williamson 2004).

Assuming that $\beta(\cdot)$ is searched within a reproducing kernel Hilbert space $\mathcal{H}$, $\langle \beta(\cdot), K(\cdot, x') \rangle = \beta(x')$ is true from its reproducing property. Let $E_i(\beta)$ be the regularized empirical error for $x_T^{(i)}$:

$$E_i(\beta) = -log\beta(x_T^{(i)}) + \frac{\lambda}{2} \|\beta\|_{\mathcal{H}}^2 \tag{4}$$

where $\lambda(> 0)$ is the regularization term, and $\|\cdot\|_{\mathcal{H}}$ denotes the norm in $\mathcal{H}$. The density ratio $\hat{\beta}$ can be updated online using a new instance $x_T^{(N+1)}$ by,

$$\hat{\beta}' = \hat{\beta} - \eta \partial_\beta E'_{N+1}(\hat{\beta}) \tag{5}$$

where $\eta(> 0)$ is the learning rate, and ∂_β denotes partial derivative with respect to β. This implies that the parameters $\boldsymbol{\alpha}$ are updated as follows.

$$\begin{cases} \hat{\alpha}_j^t \leftarrow (1-\eta\lambda)\hat{\alpha}_{j+1}^{t-1},\ j = 1, ..., N-1 \\ \hat{\alpha}_j^t \leftarrow \frac{\eta}{\beta(\hat{x}_T^{(N+1)})},\ j = N \end{cases} \tag{6}$$

where η and λ are user-defined parameters representing the regularization term, and learning rate respectively. In MSR, we use the above formula to update $\boldsymbol{\alpha}$ every time an instance arrives in T.

MSR can be used for any learning algorithm that incorporates importance weight of training instances. As new instances continue to arrive in S and T, the regression model is updated when a concept drift is detected between the two distributions represented by the weighted source data stream and target data stream. MSR predicts the target value of each instance in T using the updated regression model.

Drift Detection Module (DDM)

The probability density function of target stream $P_T(x)$ is estimated by $\hat{P}_T = \beta(x)P_S(x)$. A drift is detected if there is a significant difference between $P_T(x)$ and $\beta(x)P_S(x)$. Let $\boldsymbol{\alpha}^0$ be the set of initial parameters. These parameters are updated online as new instances arrive in S and T. Let $\boldsymbol{\alpha}^t$ represent the parameters at time t. The difference between the distributions is quantized by taking the likelihood ratio. A drift is detected if it is more than a user-defined threshold μ, as follows.

$$S = \sum_{j=1}^{N} \ln \frac{P_T(x_T^{(j)})}{\hat{\beta}_0 P_S(x_T^{(j)})} = \sum_{j=1}^{N} \ln \frac{\hat{\beta}_t(x_T^{(j)})}{\hat{\beta}_0(x_T^{(j)})} > \mu \tag{7}$$

Algorithm 2 Drift Detection.

```
1: Set $t = 0$;
2: Run Density Ratio Estimation (Algorithm 1) to get initial set of parameter $\boldsymbol{\alpha}^0$;
3: while not at the end of streams do
4:     $t \leftarrow t + 1$;
5:     Online update $\boldsymbol{\alpha}^{t-1}$ to $\boldsymbol{\alpha}^t$;
6:     Calculate change score $S$;
7:     if $S > \mu$ then
8:         Re-evaluate parameter set $\boldsymbol{\alpha}^0$;
9:         $\hat{\beta}(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j^t K_\sigma(\boldsymbol{x}, x_T^{(j)}), \boldsymbol{x} = \{x_S^{(i)}\}_{i=1}^{N}$;
10:        Train regression model using Eq.1 .
11:    end if
12: end while
```

where $\hat{\beta}_0$ and $\hat{\beta}_t$ are density ratios defined by $\boldsymbol{\alpha}^0$ and $\boldsymbol{\alpha}^t$ respectively.

Regression Module

Initially, we train a regression model using a small set of data instances from both S and T. Here, we estimate the importance weight for each source data instance using DRM to overcome covariate shift between the two streams.

The regression model needs to be updated over time if there is any asynchronous concept drift in the source or target stream. Algorithm 2 shows the drift detection and parameter update process in MSR. As new data instances arrive in S and T, concept drifts are detected by DDM. After detecting a drift, we compute $\beta(\cdot)$ for instances in $\mathbf{W}_S$ using the updated $\boldsymbol{\alpha}$ from DRM in Line 9. Then, we train a new regression model according to Eq. 1 using weighted $\mathbf{W}_S$ and $\mathbf{W}_T$ in Line 10.

Theoretical Analysis

We now derive the convergence rate of the proposed solution, and its computational complexity.

Convergence Rate

In order to derive convergence rate, we first prove that the empirical error $E_i(\beta)$ is a strongly convex function over β (lemma 1). Then, we use induction (in lemmas 2-3) to obtain the convergence rate for updating $\boldsymbol{\alpha}$.

Lemma 1 *$E_i(\beta)$ is strongly convex function, i.e., $\forall\lambda > 0$, $E_i(\beta) - \lambda\|\beta\|^2$ is convex.*

Proof 1 *In Eq. 4, the log term is concave by definition, and hence the negative log is convex. Combining the regularization term and $\lambda\|\beta\|^2$, the result is a convex function. Using the additive property of convexity, $E_i(\beta)$ is strongly convex.*

From Lemma 1, we know $E_i(\beta)$ is λ-strongly convex. We assume $E_i(\beta)$ is μ-smooth in the neighborhood of (optimum) β^*, which means for all $\beta \in \boldsymbol{\beta}$ (where $\boldsymbol{\beta}$ is a set of all instance weights), the following inequality holds (Ying 2007).

$$\mathbb{E}[E_i(\beta) - E_i(\beta^*)] \leq \frac{\mu}{2}\mathbb{E}[\|\beta - \beta^*\|^2] \tag{8}$$

Lemma 2 *Let f_1 be an arbitrary point on $\partial_\beta E_i(\beta)$. $\hat{f}_1$ be the gradient at f_1 and β_1 its corresponding importance weight. For a constant $G > 0$, if $\mathbb{E}[\|\hat{f}_1\|^2] \leq G^2$, then $\mathbb{E}[\|\beta_1 - \beta^*\|^2] \leq \frac{4G^2}{\lambda^2}$.*

Proof 2 *Due to strong convexity of $E_i(\beta)$,*

$$\langle f_1, \beta_1 - \beta^* \rangle \geq \frac{\lambda}{2}\|\beta_1 - \beta^*\|^2 \tag{9}$$

where $\langle \cdot, \cdot \rangle$ denotes a dot product. Based on the Cauchy-Schwartz inequality,

$$\|f_1\|^2 \geq \frac{\lambda^2}{4}\|\beta_1 - \beta^*\|^2 \tag{10}$$

and

$$\mathbb{E}\left[\|\hat{f}_1\|^2\right] = \mathbb{E}\left[\|f_1 + (\hat{f}_1 - f_1)\|^2\right] \geq \mathbb{E}[\|f_1\|^2] \tag{11}$$

Based on the inequality in Eqs. 9, 10 and 11, we get

$$\mathbb{E}\left[\|\beta_1 - \beta^*\|^2\right] \leq \frac{4}{\lambda^2}\mathbb{E}\left[\|\hat{f}_1\|^2\right] \leq \frac{4G^2}{\lambda^2} \tag{12}$$

Lemma 3 *Let $E_i(\beta)$ be λ-strongly convex over the convex set $\beta \in \boldsymbol{\beta}$, and $\mathbb{E}[\|\hat{f}_n\|^2] \leq G^2$, where G is a constant. Then, with a learning rate $\eta_n = \frac{1}{\lambda n}$, the following inequality holds,*

$$\mathbb{E}\left[\|\beta_n - \beta^*\|^2\right] \leq \frac{4G^2}{\lambda^2 n} \tag{13}$$

Proof 3 *We prove by induction. Lemma 2 provides the base case. Since $E_i(\beta)$ is λ-strongly convex, we have*

$$\langle f_n, \beta_n - \beta^* \rangle \geq E(\beta_n) - E(\beta^*) + \frac{\lambda}{2}\|\beta_n - \beta^*\|^2 \tag{14}$$

and,

$$E(\beta_n) - E(\beta^*) \geq \frac{\lambda}{2}\|\beta_n - \beta^*\|^2 \tag{15}$$

Let us assume $\mathbb{E}[\|\beta_n - \beta^*\|^2] \leq \frac{4G^2}{n\lambda^2}$. For any point V and $\beta \in \boldsymbol{\beta}$, $\|\prod_{\boldsymbol{\beta}}(V) - \beta\| \leq \|V - \beta\|$. Based on the inequality in Eqs. 14 and 15,

$$\begin{aligned}
\mathbb{E}\left[\|\beta_{n+1} - \beta^*\|^2\right] &= \mathbb{E}\left[\left\|\prod_{\boldsymbol{\beta}}(\beta_n - \eta_n \hat{f}_n) - \beta^*\right\|^2\right] \\
&\leq \mathbb{E}\left[\|\beta_n - \eta_n \hat{f}_n - \beta^*\|^2\right] \\
&\leq \mathbb{E}\left[\|\beta_n - \beta^*\|^2\right] - 2\eta_n \mathbb{E}\left[\lambda\|\beta_n - \beta^*\|^2\right] + \eta_n^2 G^2 \\
&= (1 - 2\eta_n\lambda)\mathbb{E}\left[\|\beta_n - \beta^*\|^2\right] + \eta_n^2 G^2 \\
&\leq \left(\frac{n}{n+1} - \frac{1}{n}\right)\frac{4G^2}{n\lambda^2} + \frac{G^2}{n^2\lambda^2} \\
&\leq \frac{4G^2}{(n+1)\lambda^2}
\end{aligned} \tag{16}$$

Therefore,

$$\mathbb{E}\left[\|\beta_n - \beta^*\|^2\right] \leq \frac{4G^2}{n\lambda^2} \tag{17}$$

So the convergence rate for updating $\boldsymbol{\alpha}$ is $\mathcal{O}\left(\frac{1}{N}\right)$.

data set	Feature	Size
PowerConsumption	4	97,966
CASP	9	45,603
AirlineDelay	6	91672
SynGlobalAbrupt	5	100,975
SynGlobalGradual	5	102,162
SynLocalAbrupt	5	101,486

Table 1: Characteristics of data sets

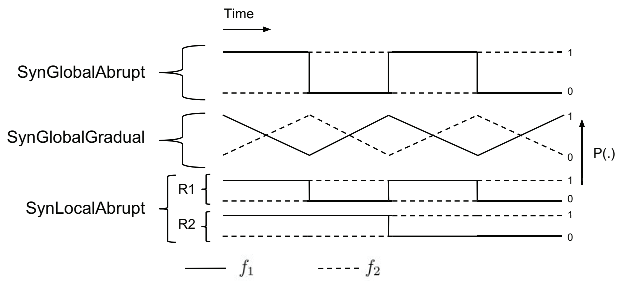


Figure 2: Illustration of synthetic data generation. Here, $P(\cdot)$ represents sampling probability.

In MSR, we perform online updates with batch training whenever a change point is detected. For batch training, the convergence rate is $\mathcal{O}(\frac{1}{N})$. Combining Lemma 3 with batch update, the overall convergence rate is $\mathcal{O}(\frac{1}{N})$.

Complexity Analysis

DRM learns $\boldsymbol{\alpha}$ only when a change is detected. This mainly involves 2 operations. One is to learn $\boldsymbol{\alpha}$ (according to Algorithm 1), and the other is to update them (according to Eq. 6). It takes $O(N^2)$ time to learn $\boldsymbol{\alpha}$, and $O(N)$ to update $\boldsymbol{\alpha}$, where N is the size of the sliding windows. In the case of DDM, the time complexity is $O(N)$ as described in Algorithm 2. For linear regression, the time complexity is $O(v^2N)$, where v is feature dimension. Therefore, the total time complexity of MSR is $O(N^2)$.

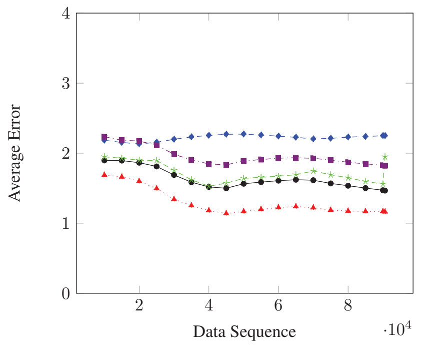
The space complexity of DRM is $O(N^2)$, dominating the space complexity of other modules. Therefore, the overall space complexity of MSR is $O(N^2)$. Moreover, both the time and space complexity of MSR are functions of N. In real world applications, N can be adjusted to execute the MSR with available resource.

Empirical Evaluation

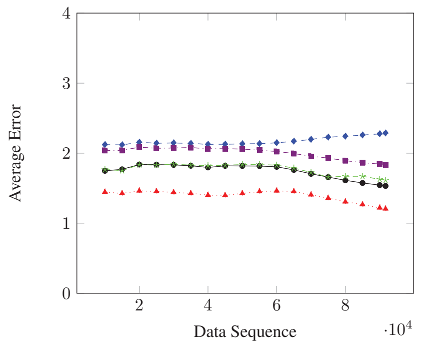
Data Sets

We use 3 real-world and 3 synthetic data sets to evaluate the proposed framework. Table 1 lists them with corresponding properties.

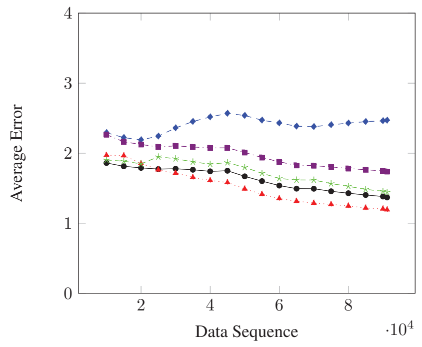
Real-World Data Sets The task in *PowerConsumption* (Lichman 2013) data set is to predict the total power to be consumed by households in 2006 from readings such as reactive and active power, voltage, and intensity. In *CASP* ((Lichman 2013)) data set, the task is to predict the size of residue, given physiochemical properties of protein tertiary



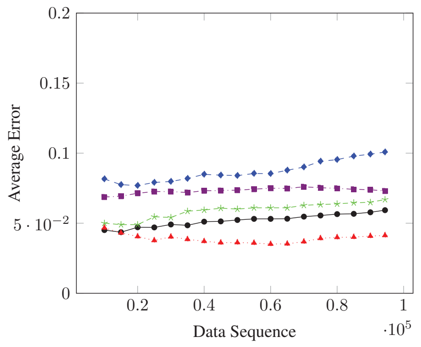
(a) *SynGlobalAbrupt*



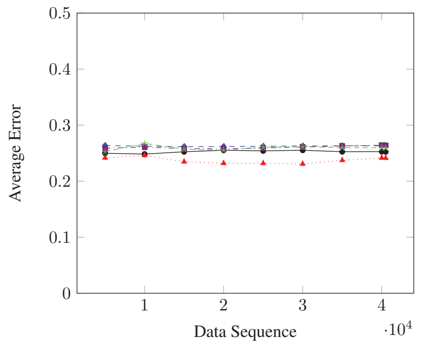
(b) *SynGlobalGradual*



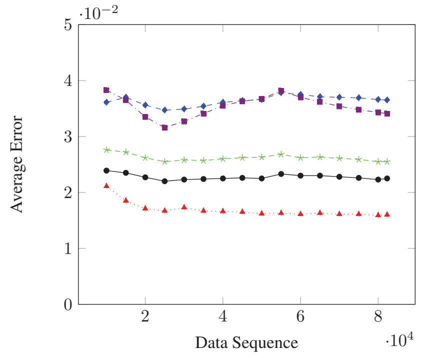
(c) *SynLocalAbrupt*



(d) *PowerConsumption*



(e) *CASP*



(f) *AirlineDelay*

Figure 3: Average Error: ● MSR; ◆ LR; ■ uLR; ✳ uMSR; ▲ tLR

structure. Finally, the task in *AirlineDelay* (Data Expo 2009) data set is to predict the arrival time delay of flights using features such as scheduled departure and arrival time, departure delay, and distance. Particularly the original AirlineDelay data set contains nearly 120M records. For simplicity, we choose to use data only from a single airline (chosen uniformly at random), totaling about 100k records.

Following the strategy in (Kurlej and Woźniak 2011), we simulate concept drifts in these data sets as follows. We first normalize all features within the range of $[0, 1]$. Then, at a time instance, which is probabilistically selected between every 1000 to 2000 instances, we choose two features at random. We then rotate them by 90 degrees clockwise, i.e., if $X_1 \times X_2$-space is selected, then $(x_1, x_2) \xrightarrow{R(\frac{\pi}{2})} (-x_2, x_1)$.

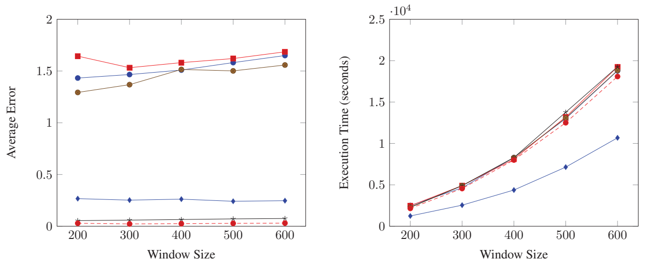
Synthetic Data sets We create the *SynX* data sets with induced concept drift using a method similar to (Ikonomovska et al. 2009). Here, X is either GlobalAbrupt, GlobalGradual or LocalAbrupt.

We synthesize the data in a 5 dimensional unit hypercube. Initially, we define two constrained regions in the hypercube. The first region (denoted by $R1$) is with $0 < x_0 < 1$, $x_1 < 0.3$, $x_2 < 0.3$, $x_3 > 0.7$, and $x_4 < 0.3$, and the second region (denoted by $R2$) is $0 < x_0 < 1$, $x_1 > 0.7$, $x_2 > 0.7$, $x_3 < 0.3$, and $x_4 > 0.7$. We sample instances from these regions for generating synthetic data sets. Furthermore, we use two different target functions for inducing concept drift. These functions are as follows.

$$f_1(x) = 10sin(\pi x_0 x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4 + \sigma(0,1)$$
$$f_2(x) = 10sin(\pi x_3 x_4) + 20(x_1 - 0.5)^2 + 10x_0 + 5x_2 + \sigma(0,1)$$

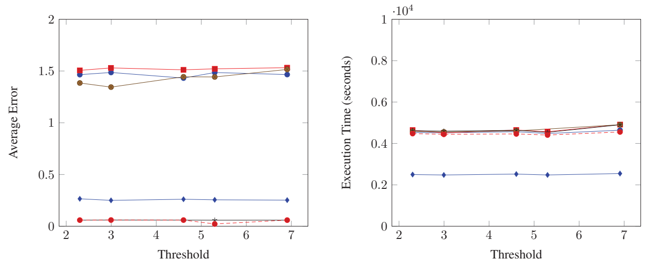
Particularly, we sample data instances from one function for a short period of time, and then switch to sample from a different function. This switch to induce concept drift is illustrated in Figure 2. For generating *SynGlobalAbrupt* and *SynGlobalGradual* data sets, we sample data from both the regions. But, they differ in the way the target functions are utilized. In the SynGlobalAbrupt, we abruptly switch sampling process from one target function to another after every 1000 to 2000 instances. This simulates the concept drift, similar to the real-world data set. Whereas, in SynGlobalGradual, we first gradually (probabilistically) increase the sampling of instances using one function, while decrease the sampling from the other function.

In the case of *SynLocalAbrupt* data set, we sample instances from both regions, but only change the target function within a one region at a time. Concretely, we use the target function f_1 in both the regions to generate data. At the first change point, we replace f_1 with f_2 in the first region only, while maintaining f_1 in the second region. At the next change point, we replace f_1 with f_2 for the first region, and replace f_2 with f_1 for the second region. Finally, at the last change point, we replace f_2 with f_1 for the first region and retain f_2 for the second region.



(a) Average Error (b) Execution Time

Figure 4: Sensitivity of MSR to difference window size (N): —●— *SynGlobalAbrupt*; —■— *SynGlobalGradual*; —●— *SynLocalAbrupt*; —♦— *CASP*; —⋆— *PowerConsumption*; - -●- - *AirlineDelay*



(a) Average Error (b) Execution Time

Figure 5: Sensitivity of MSR to difference threshold (μ): —●— *SynGlobalAbrupt*; —■— *SynGlobalGradual*; —●— *SynLocalAbrupt*; —♦— *CASP*; —⋆— *PowerConsumption*; - -●- - *AirlineDelay*

Multistream We generate a biased source stream in each data set following (Sugiyama et al. 2008). We randomly choose one sample from the data pool between two consecutive change points, and accept this as source sample with probability $\min(1, 4(x_i^d))$, where x_i^d is the d-th feature of $\mathbf{x}_i$; then we remove $\mathbf{x}_i$ from the pool regardless of its rejection or acceptance. Once 10% of data instances are selected as source, we choose the rest as target. Here, d is randomly determined for real-world dataset. However, we fix $d = 0$ for synthetic data set since only x_i^0 is located in the region $[0, 1]$. Finally, we concatenate the source and target data to simulate the respective streams.

Experiments

Baseline Since no previous methods on multistream regression exist, we devise four baseline methods based on our problem framework.

The first baseline method, we call *LR*, trains a simple weighted linear regression model on the initial set of source instances and then predicts the target values of data instances in T. For the second baseline method, we extend the simple linear regression model by periodically re-training on the latest data instances occurring in S after every 1000 target instances. This method is denoted by *updated-LR* or *uLR*. Next, we train the third baseline regression method directly on the target stream and update every 1000 instances. We denote it by *target-LR* or *tLR*. Note that this supervised periodic training uses the true response value at the target stream, and therefore should result in the least average error among all competing methods. Finally, to demonstrate the effect of covariate shift correction, we employ MSR without the drift detection module. Instead, we periodically update the regression model after every 1000 instances, similar to uLR and tLR. We call this baseline method as *update-MSR* or *uMSR*.

Setup The MSR approach involves multiple parameters. We use $N = 300$ as our default setting in the experiments. Also, $\lambda = 0.01$ and $\eta = 1$ following (Kawahara and Sugiyama 2012).

Result

Performance Figure 3 shows the progress of average error as instances arrive in the target stream T. The average error of MSR asymptotically outperforms all competing methods (except tLR) on each data sets. Since tLR uses the truth values directly on target instances, it achieves the highest performance. Moreover, uLR has a much lower error on all data sets compared to LR. This indicates that the regression model needs to be updated to overcome concept drift. Yet, a better performance of MSR indicates that drift detection provides a better frequency of updates rather than simple periodic updates. Moreover, uMSR performs better than uLR despite them having the same periodic update frequency. This demonstrates the existence of covariate shift between the source and target streams, and also the effect of correcting the same.

Sensitivity In the next set of experiments, we measure parameter sensitivity of the proposed approach. First, we study the sensitivity of MSR to window size or N. Figure 4 shows the average error and execution time when using different values of N. It can be observed that MSR is marginally sensitive to the window size with respect to average error. However, the execution time increases with N since the time complexity of MSR depends on the window size. It should be noted that the execution time of MSR is greater than uLR or tLR due to DRM and DDM. Nevertheless, MSR achieves greater prediction performance at the cost of execution time.

Figure 5 shows sensitivity of MSR to different threshold (μ) values for all the data sets. It can be observed that changing the drift detection threshold does not affect the average error and execution time significantly. Overall the experiment results indicate that MSR is not too much sensitive to the parameters.

Conclusion

In this paper, we perform regression in the multistream setting involving two independent, yet related data streams. We address the main challenges of covariate shift and asynchronous concept drift simultaneously by estimating probability density ratio using a Gaussian kernel model that utilizes available truth value on the source stream to predict the response-variable value on the target stream. Particularly, we present an online mechanism to update its parameters, and utilize them for drift detection. We also study its theoretical properties. Our empirical evaluation on real-world and synthetic data sets demonstrate that our method performs significantly better than the baseline approaches.

Acknowledgments

We thank the reviewers for their insightful comments. This material is based upon work supported by NSF award number DMS-1737978, AFOSR award number FA9550-14-1-0173, and IBM faculty award (Research).

References

Chandra, S.; Haque, A.; Khan, L.; and Aggarwal, C. 2016. An adaptive framework for multistream classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, 1181–1190. New York, NY, USA: ACM.

Chen, X.; Monfort, M.; Liu, A.; and Ziebart, B. D. 2016. Robust covariate shift regression. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, 1270–1279.

Data Expo. 2009. ASA sections on statistical computing and statistical graphics. http://stat-computing.org/dataexpo/2009/.

Fan, W.; an Huang, Y.; Wang, H.; and Yu, P. S. 2004. Active mining of data streams. In *in Proceedings of the Fourth SIAM International Conference on Data Mining*, 457–461.

Gama, J.; Medas, P.; Castillo, G.; and Rodrigues, P. 2004. Learning with drift detection. In *In SBIA Brazilian Symposium on Artificial Intelligence*, 286–295. Springer Verlag.

Gama, J. a.; Žliobaitė, I.; Bifet, A.; Pechenizkiy, M.; and Bouchachia, A. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46(4):44:1–44:37.

Haque, A.; Khan, L.; Baron, M.; Thuraisingham, B. M.; and Aggarwal, C. C. 2016. Efficient handling of concept drift and concept evolution over stream data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, 481–492.

Haque, A.; Wang, Z.; Chandra, S.; Dong, B.; Khan, L.; and Hamlen, K. W. 2017. Fusion: An online method for multistream classification. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, 919–928. New York, NY, USA: ACM.

Haque, A.; Khan, L.; and Baron, M. 2016. SAND: semi-supervised adaptive novel class detection and classification over data stream. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 1652–1658.

Huang, J.; Gretton, A.; Borgwardt, K. M.; Schölkopf, P. B.; and Smola, A. J. 2007. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19*. MIT Press. 601–608.

Ikonomovska, E., and Gama, J. 2008. Learning model trees from data streams. In *Discovery Science, 11th International Conference, DS 2008, Budapest, Hungary, October 13-16, 2008. Proceedings*, 52–63.

Ikonomovska, E.; Gama, J.; Sebastião, R.; and Gjorgjevik, D. 2009. *Regression Trees from Data Streams with Drift Detection*. Springer Berlin Heidelberg. 121–135.

Kawahara, Y., and Sugiyama, M. 2012. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining* 5(2):114–127.

Kivinen, J.; Smola, A. J.; and Williamson, R. C. 2004. Online learning with kernels. *IEEE Trans. Signal Processing* 52(8):2165–2176.

Kurlej, B., and Woźniak, M. 2011. *Learning Curve in Concept Drift While Using Active Learning Paradigm*. Springer Berlin Heidelberg. 98–106.

Lichman, M. 2013. UCI machine learning repository.

Masud, M. M.; Al-Khateeb, T.; Khan, L.; Aggarwal, C. C.; Gao, J.; Han, J.; and Thuraisingham, B. M. 2011. Detecting recurring and novel classes in concept-drifting data streams. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, 1176–1181.

Nadungodage, C. H.; Xia, Y.; Vaidya, P. S.; Chen, Y.; and Lee, J. J. 2014. Online multi-dimensional regression analysis on concept-drifting data streams. *International Journal of Data Mining, Modeling and Management* 6(3):217–238.

Natrella, M. 2010. Nist/sematech e-handbook of statistical methods.

Rosenthal, F.; Volk, P. B.; Hahmann, M.; Habich, D.; and Lehner, W. 2009. Drift-aware ensemble regression. In *Machine Learning and Data Mining in Pattern Recognition, 6th International Conference, MLDM 2009, Leipzig, Germany, July 23-25, 2009. Proceedings*, 221–235.

Sugiyama, M.; Nakajima, S.; Kashima, H.; von Bünau, P.; and Kawanabe, M. 2007. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, 1433–1440.

Sugiyama, M.; Suzuki, T.; Nakajima, S.; Kashima, H.; von Bünau, P.; and Kawanabe, M. 2008. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics* 60(4):699–746.

Ying, Y. 2007. Convergence analysis of online algorithms. *Adv. Comput. Math.* 27(3):273–291.

Zadrozny, B. Z. 2004. Learning and evaluating classifiers under sample selection bias. In *International Conference on Machine Learning (ICML)*, 903–910.