Adaptive Model Tree for Streaming Data

Anca M. Zimmer*, Michael Kurze*, Thomas Seidl*,
* *RWTH Aachen University, {anca.zimmer, michael.kurze, seidl}@cs.rwth-aachen.de*

Abstract

With an ever-growing availability of data streams the interest in and need for efficient techniques dealing with such data increases. A major challenge in this context is the accurate online prediction of continuous values in the presence of concept drift. In this paper, we introduce a new adaptive model tree (AMT), designed to incrementally learn from the data stream, adapt to the changes, and to perform real time accurate predictions at anytime. To deal with submodels lying in different subspaces, we propose a new model clustering algorithm able to identify subspace models, and use it for computing splits in the input space. Compared to state of the art, our AMT allows for oblique splits, delivering more compact and accurate models.

1. Introduction

Prediction of continuous values is an important task in many applications, e.g. financial, scientific. Generalizing a set of observations from a process provides a better insight, reveals trends, and allows for accurate prediction. Many applications used nowadays involve streaming data, i.e. the data arrives continuously, is therefore extremely large, and the underlying model changes with time. Since it becomes unfeasible to store the samples and process them offline, the samples must be processed as they arrive, and then discarded. As the model becomes obsolete with time, the detection of changes in the underlying generating mechanism is important. The model must incorporate the arriving samples, continuously adapt, and in the same time enabling accurate predictions in real-time.

Closely related to decision trees, model (or regression) trees offer an interpretable model of the data and enable accurate predictions of continuous values. In each inner node they contain a test in the input space, and in each leaf a linear model predicting the output. While several batch algorithms for constructing model trees have been proposed in the literature, the literature regarding incremental model trees is limited.

The first model tree, CART, introduced in [4], has axis parallel splits, the standard deviation of the predictor as impurity measure, and the sample means as constant models in the leaves. In [14] linear regression models are used in the leaves, and later the residual sum of squares w.r.t. the linear model was used as impurity measure. In [5] a regression tree with oblique splits was introduced. The first step towards an incremental construction of regression trees was made in [12], but its increasing update runtime with the number of training samples, and its inability to deal with concept drifts, make it unfeasible for streaming data. In [11] a regression tree is proposed, which is able to deal with data streams with unknown dynamics. The algorithm focuses on determining an adaptive window of samples to be stored, while the concerns regarding the tree construction are of minor matter, axis parallel splits being made with k-Means clustering. Ikonomovska et al. introduced in [6] a fast incremental model tree (FIMT) for learning from time-changing data streams. FIMT uses perceptrons as leaf models, the standard deviation of the predictor as impurity measure, and computes only axis parallel splits. Since data perfectly described by a linear model may still have a high standard deviation of the predictor, this impurity measure leads to unnecessary many splits. This effect is also enhanced by the axis parallel splits.

In this paper we introduce a new adaptive model tree (AMT) for streaming data. The split strategy employs clustering for the split decision, and training a linear classifier to define the split boundary. We propose a new model clustering, able to identify subspace models and use an impurity measure consistent with the model. Since we do not restrict to axis parallel splits, but allow oblique splits, we achieve higher prediction accuracy. We show that AMT delivers efficient, outperforming the state of the art FIMT.

After discussing possible leaf models in Section 2, a new model clustering algorithm is proposed in Section 3, which is used for the split strategy of AMT. Finally, in Section 4 we present the AMT, followed by a detailed evaluation on both synthetic and real datasets in Section 5, and the conclusion in Section 6.

2. Leaf Models

Throughout this paper we assume observations of the form $(\mathbf{x}, y)$, with input $\mathbf{x} = [x_1, ..., x_d]$ and output $y \in \mathbb{R}$. For a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the matrix containing in each row the inputs of a sample, and $\mathbf{y} \in \mathbb{R}^n$ the vector containing the outputs of the observations. W.l.o.g. we consider z-score normalized data. In this Section we consider three different approaches to build a model from $\mathcal{D}$.

Ordinary Least Squares (OLS). The most popular regression technique is OLS, which computes the regression coefficients β as:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{1}$$

If we consider the Cholesky decomposition $\mathbf{X}^T\mathbf{X} = \mathbf{L}\mathbf{L}^T$, then β can be solved efficiently in $O(d^2)$. The complexity of the Cholesky decomposition is $O(d^3)$, and can be incrementally updated in $O(d^2)$ by using a rank-one update procedure from [9]. Hence, β can be incrementally updated with each new sample in $O(d^2)$.

Ridge Regression (RR). RR is a variant of linear regression in which unimportant coefficients shrink towards zero and linear models are detected in subspaces. The coefficients β are computed as:

$$\beta = (\mathbf{X}^T\mathbf{X} + \lambda_R\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2}$$

where $\mathbf{I} \in \mathbb{R}^{d \times d}$ is the identity matrix, and $\lambda_R \in \mathbb{R}$ a regularization factor. The residuals' variance decreases for an increasing λ_R, while the bias increases. Incremental updates are similarly computed to OLS.

Partial Least Squares (PLS). Similar to ridge regression, PLS [15] is a rank reducing method which generates robust regression models for noisy data. The main idea is to perform dimensionality reduction in the input space such that the covariance to the output remains preserved, and then perform the regression in this lower dimensional space. PLS performs following decomposition $\mathbf{X} = \mathbf{T}\mathbf{P}^T + E$, where $\mathbf{P} \in \mathbb{R}^{d \times r}$ is a projection (loading) matrix, reducing the input dimensionality to $r < d$, and $\mathbf{T} \in \mathbb{R}^{n \times r}$ is the dimensionality reduced score matrix. Then, the relationship between $\mathbf{X}$ and $\mathbf{y}$ is modeled via the linear equation: $\mathbf{y} = \mathbf{T}\mathbf{b}$, with $\mathbf{b} \in \mathbb{R}^r$. Since $\mathbf{T} = \mathbf{X}\mathbf{P}$, it follows:

$$\mathbf{Y} = \mathbf{X}\mathbf{P}\mathbf{b} = \mathbf{X}\beta, \text{ with } \beta = \mathbf{P}\mathbf{b}, \beta \in \mathbb{R}^d.$$

While the bias-variance trade off for ridge regression is controlled by λ_R, for PLS it is controlled by the number r of score vectors. Because individual scores are orthogonal, the unbiased OLS solution is obtained by using the full set of d scores. We choose an appropriate

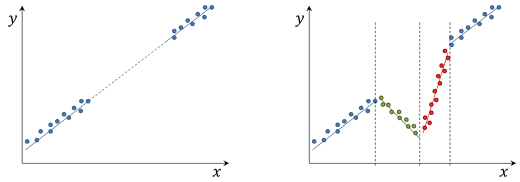

Figure 1. Cluster separability in input space

$r < d$ such that at least a given fraction $\varphi \in [0, 1]$ of the explainable output variance is covered:

$$r = \min_{1 \le i \le d} \frac{\sigma_0^2 - \sigma_e^2}{\sigma_0^2 - \sigma_d^2} \ge \varphi, \tag{3}$$

where σ_0 is the variance in the output dimension, and σ_e is the variance of the output after the eth iteration.

As shown in [10], [13], it is possible to update the regression coefficients computed by PLS at the arrival of a new sample, without the previous samples in $O(d^2)$.

Impurity Measure Since we use as leaf models linear regression models, obtained by minimizing the *mean squared error* (mse), this is also used as the impurity measure, which assesses the quality of a split. For monitoring the performance in a streaming context, we use the *prequential squared error* (pse) with a fading mechanism as discussed in [8], with a fading factor γ. The smaller γ is, the sooner old errors are forgotten.

3. Split Strategy

The goal of a split is to partition the data, such that the emerging models fit the data as well as possible, i.e. the sum of the impurities of the emerging models is as small as possible. With a good split strategy, less splits are necessary. We propose a new model clustering algorithm to obtain a good partitioning of the data, and then generalize the split in the input space by training a linear classifier.

Subspace Model Clustering. We propose in this paper a new agglomerative regression model clustering (MoClus), specifically designed to fit the requirements of a model tree. Like any other agglomerative clustering it starts with a high number of clusters, and progressively merges them. Two main components are required: a measure for assessing the quality of a cluster, and a strategy for assigning points to clusters. Since the goal is to obtain clusters in which the output is as well as possible described by a linear model, we propose to describe a cluster by a model $\mathcal{M} \in \{\text{OLS, RR, PLS}\}$ and asses its quality by the mse

as impurity measure. W.r.t. the assigning strategy, the cluster separability in the input space must be ensured. This is an important aspect of model trees, since at the arrival of a new sample first the corresponding partition in the input space must be determined, then the corresponding model applied. We illustrate an example in Figure 1: while in the left image the blue points all belong to the same cluster, in the right image they must be splitted, because of the red and green cluster lying in between.

MoClus, summarized in Algorithm 1, takes the number of random seeds k_0 and final number of clusters k as input parameters, together with α, which indicates how many clusters are merged in each iteration. $\mathcal{T}$ is the training set, and $\mathcal{M}$ the type of model used for the clusters. In the initialization k_0 seeds are randomly chosen, each sample $\mathbf{o}_l \in \mathcal{T}$ is assigned to the closest seed, according to the L_2 norm in $\mathbb{R}^{d+1}$, and the mean for each cluster $\mathcal{C}_i$ in $\mathbb{R}^{d+1}$ is used as a constant cluster model f_i (cf. line 1-4). Then, several iterations are computed, until the desired number of clusters is reached. In each iteration samples are assigned to clusters (cf. line 7-15), and then clusters are merged (cf. line 16). The assignment happens in two steps. First, each sample $(\mathbf{x}, y) \in \mathcal{T}$ is assigned to the cluster $\mathcal{C}_i$ which best predicts the output: $\underset{1\leq i\leq k_c}{min}\ (f_i(\mathbf{x}) - y)^2$. Then, the projection $\mathcal{IC}_i$ of each cluster $\mathcal{C}_i$ onto the input space is computed, and a second assignment takes place in the input space. A linear separation of the clusters $\mathcal{IC}_i$ is performed with LDA, and each object is assigned to a cluster according to the decision function δ_{LDA} (line 14). Finally, a model is learned for each newly built cluster (line 15). This ensures that there are no big overlaps between the clusters in the input space, and a linear separator can be better trained. In each iteration the number of clusters is reduced by a factor of $\alpha \in (0, 1)$. Algorithm 2 describes how the number of clusters is reduced from k to k' in one iteration. All possible cluster merges $\mathcal{C}_{ij}$ are computed, together with the corresponding prediction model f_{ij}. Among these possibilities the clusters merges with the lowest mse$(f_{ij}, \mathcal{C}_{ij})$ are actually performed.

Note that the choice of $\mathcal{M}$ influences whether MoClus is a full space or a subspace clustering algorithm. If $\mathcal{M} \in \{$PLS,RR$\}$, then MoClus is a subspace clustering. In this case, MoClus is related to the subspace correlation clustering algorithm ORCLUS (introduced in [1]). While ORCLUS describes the clusters by eigenvectors from the principal component analysis, and systematically reduces the cluster dimensionality in each iteration until a user defined value is reached,

Algorithm 1 MoClus

Require: $k_0 \in \mathbb{N}, k \in \mathbb{N}, \alpha \in [0, 1], \mathcal{T}, \mathcal{M}$
1: select k_0 random seeds $\{\mathbf{s} = (\mathbf{x}_i, y_i)\}_{i=0}^{k_0}$ from $\mathcal{T}$
2: **for** $i \leftarrow 1 \dots k_0$ **do**
3: $\quad \mathcal{C}_i = \{\mathbf{o} \in \mathcal{T} | L_2(\mathbf{o}, \mathbf{s}_j) > L_2(\mathbf{o}, \mathbf{s}_i), 1 \leq j \leq k_0\}\}$
4: $\quad f_i \leftarrow \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{o}_l \in \mathcal{C}_i} y_l$
5: $k_c = k_0$
6: **repeat**
7: $\quad$ assign $\mathbf{o}_l$ to $\mathcal{C}_i$: $\underset{1\leq i\leq k_c}{min}\ (f_i(\mathbf{x}_l) - y_l)^2$, $1 \leq l \leq |\mathcal{T}|$
8: $\quad$ **for** $i \leftarrow 1 \dots k_c$ **do**
9: $\quad\quad \mathcal{IC}_i$ = projection of $\mathcal{C}_i$ onto the input space
10: $\quad\quad$ compute μ_i, centroid of $\mathcal{IC}_i$
11: $\quad$ compute the common Σ for LDA
12: $\quad$ compute decision function δ_{LDA}
13: $\quad$ **for** $i \leftarrow 1 \dots k_c$ **do**
14: $\quad\quad \mathcal{C}_i = \{\mathbf{o} \in \mathcal{T} | \delta_{LDA}(\mathbf{o}) = i, 1 \leq j \leq k_c\}\}$
15: $\quad\quad f_i \leftarrow \mathcal{M}(\mathcal{C}_i)$
16: $\quad$ MERGE$(\mathcal{C}_1, ..., \mathcal{C}_{k_c}, \lfloor \alpha k_c \rfloor)$
17: $\quad k_c = \lfloor \alpha k_c \rfloor$
18: **until** $k_c = k$
Ensure: $(\mathcal{C}_1, f_1), ..., (\mathcal{C}_k, f_k)$

Algorithm 2 MoClus: merge

1: **function** MERGE$(\mathcal{C}_1, ..., \mathcal{C}_k,\ k')$
2: $\quad$ compute all $\mathcal{C}_{ij} = \mathcal{C}_i \cup \mathcal{C}_j,\ 1 \leq i \neq j \leq k$
3: $\quad$ **repeat**
4: $\quad\quad f_{ij} \leftarrow \mathcal{M}(\mathcal{C}_{ij}),\ 1 \leq i \neq j \leq k$
5: $\quad\quad r_{ij} = \text{mse}(f_{ij}, \mathcal{C}_{ij})$
6: $\quad\quad$ perform best merge $\mathcal{C}_{i'j'} : (i', j') = \underset{1\leq i,j\leq k}{min}\ r_{ij}$
7: $\quad\quad \mathcal{C}_{i'} \leftarrow \mathcal{C}_{i'j'}, \quad \mathcal{C}_{j'} \leftarrow \emptyset$
8: $\quad\quad$ update all $\mathcal{C}_{i'j}$ and $\mathcal{C}_{ij'},\ 1 \leq i, j \leq k$
9: $\quad\quad k = k - 1$
10: $\quad$ **until** $k = k'$
11: $\quad$ **return** $(\mathcal{C}_1, f_1), ..., (\mathcal{C}_{k'}, f_{k'})$

MoClus describes the clusters by a regression model and determines the intrinsic dimensionality of each cluster independently, from the training data. Also MoClus has the 2-steps assignment procedure, ensuring a better cluster separability in the input space. The computational complexity of MoClus depends on k_0 and α. In one merge step $\frac{k_c(k_c-1)}{2}$ cluster pairs are computed and their models built. The complexity of building models is $O(d^3 + nd^2)$ for OLS and RR, and $O(nd^2)$ for PLS. Since each sample belongs to a single cluster and is thus maximally involved in $k_c - 1$ pair computations, the cost for building the models for $\frac{k_c(k_c-1)}{2}$ pair clusters is $O\big(k_c(d^3 + nd^2)\big)$, which is the complexity of Algorithm 2. MoClus performs $\lceil \log_\alpha \frac{k}{k_0} \rceil$ merging steps, hence the its total complexity is $O\big((d^3 + nd^2) \log_\alpha^2 \frac{k}{k_0}\big)$.

Classification. After partitioning the training data, we generalize the split in the input space by training a linear classifier. Now any incoming test sample can be assigned to the corresponding leaf, whose model is then used for predicting the output. There are two common approaches for defining a linear boundary between two classes which we consider for AMT: linear discriminant analysis (LDA) and support vector machines (SVM). Both classifiers allow for incremental updates. While for LDA this amounts to updating the normal distributions, for SVM an approach was proposed in [3].

Stopping Criterion. In order to finish the recursive splitting, a stopping criterion is required. We test whether a split of $\mathcal{T}$ in $\mathcal{T}_1$ and $\mathcal{T}_2$ is meaningful or not, by checking its relative error reduction:

$$\frac{\frac{|\mathcal{T}_1|}{|\mathcal{T}|}\text{mse}(f_1,\mathcal{C}_1)+\frac{|\mathcal{T}_2|}{|\mathcal{T}|}\text{mse}(f_2,\mathcal{C}_2)}{\text{mse}(f,\mathcal{T})} \leq \tau_{red}, \quad (4)$$

where $f = \mathcal{M}(\mathcal{T})$ and $f_i = \mathcal{M}(\mathcal{T}_i)$, $i \in \{1,2\}$. Choosing a right value for τ_{red} is a compromise between accuracy and overfitting.

4. Adaptive Model Tree

After introducing the single tree components, we describe in this section how these interact in the adaptive model tree (AMT) with an incoming data stream. The adaption of the tree is two-sided: on the one hand, AMT adapts with the continuous arrival of samples generated by the same underlying function and increases its prediction accuracy, and on the other hand, it detects drifts in the underlying function and adapts itself to it. The AMT is built as a binary model tree: the training data is partitioned by MoClus in two clusters, and a linear split is trained. This procedure is recursively repeated until the stop criterion is fulfilled.

At the arrival of a new training sample $(\mathbf{x}, y)$, not only the leaf model is updated, but also all splits in the inner nodes on the path. Hence, in each inner node we keep track of two splits: one in the input space, for classifying test samples, and one in the full space, for classifying training samples. In order to allow the model tree to grow online, a window of τ_n training samples is stored in each leaf. As soon as the window is full, the samples are divided into train and test set, and a split is induced. If the split turns out to be meaningful (cf. Equation 4), then two leaf nodes are created and an additional classifier learned for the full space split. Otherwise, the collected samples are discarded and the collection starts again. Besides growing, pruning plays an important role. When the available memory is nearly exhausted, the leaves with the smallest prequential squared error are removed, like in FIMT. To detect an abrupt change we employ the detection mechanism proposed in [8], also used by FIMT, based on the Page-Hinckley test.

5. Evaluation

We investigate the prediction accuracy and efficiency, in terms of runtime and memory, of the proposed AMT. We discuss the parameters of AMT and their robustness, and compare the results to the ones obtained with the state of the art FIMT. For the change detection of both AMT and FIMT we used $\epsilon_{PH} = 0.005$ and $\tau_{PH} = 8 \cdot \sigma_y$ (where σ_y is the standard deviation of the output before normalization), and a fading factor of $\gamma = 1-10^{-4}$ for the Q-statistics. If nothing else mentioned, we use prequential windows for the evaluation, every arriving sample being first used for testing then for training. We normalized the incoming data stream by using z-score and incrementally updating the mean and standard deviation. We evaluate the accuracy by means of the relative error:

$$RE(\mathcal{T}) = \frac{\text{mad}_{AMT}(\mathcal{T})}{\text{mad}_{\overline{y}}(\mathcal{T})} = \frac{\sum_{(\mathbf{x},y)\in\mathcal{T}}|y-f(\mathbf{x})|}{\sum_{(\mathbf{x},y)\in\mathcal{T}}|y-\overline{y}|},$$

which is the relative improvement of the prediction accuracy obtained with AMT over just using the average in the output dimension. Hence, a RE of 1 means that there is no improvement over the naive approach, and a RE below 1 corresponds to an improvement.

We used following real-world and artificial data sets in our experiments:

Data Set	**Type**	d	$\lvert\mathcal{T}\rvert$
Hinge Model	artificial	2	6,561
2D Planes [4]		10	40,768
Fried [7]		10	40,768
MV Delve [6]		7	40,967
Abalone	real world [2]	8	4,377
Concrete	*(medium)*	8	1,030
Ailerons		40	13,750
Elevators		18	16,599
California Housing		8	20,500
House8L		8	22,784
House16H		16	22,784
Pole Telecom		48	15,600
Wind		14	6,574
Wine Quality		11	5,298
Power Cons.	real world	6	2,075,260
PSP	*(big)*	60	2,338,122

All evaluation experiments were conducted using a single-threaded Java-based implementation, and were run on Intel Core-2 Duo with 3 GHz and with 4G memory available to each process.

Table 1. Summary of the AMT parameters, and the standard deviation of the RE obtained by their variation

Component	Parameter	Varied values	OLS		RR		PLS	
			LDA	SVM	LDA	SVM	LDA	SVM
stop criteria	$\tau_n \in \mathbb{N}$	$\tau_n \in \{64, 75, 150, 250, \mathbf{375}, 500, 750, 1000, 1500, 2000\}$	0.020	0.022	0.020	0.017	0.028	0.033
	$0 \ll \tau_{red} \ll 10$	$\tau_{red} \in \{0.5, 0.75, 1, \mathbf{1.2}, 1.5, 1.75, 2\}$	0.023	0.014	0.018	0.015	0.027	0.028
MoClus	$2 \ll k_0 \ll 1000$	$k_0 \in \{4, \mathbf{8}, 16, 32, 64\}$	0.009	0.010	0.012	0.011	0.007	0.009
	$\alpha \in]0, 1[$	$\alpha \in \{0.1, 0.2, ..., \mathbf{0.5}, .., 0.9\}$	0.011	0.011	0.007	0.009	0.013	0.009
	λ_Σ	$\lambda_\Sigma \in \{10^{-6}, \mathbf{10}^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$	0.049	0.048	0.075	0.067	0.050	0.048
PLS	$\varphi \in]0, 1[$	$\varphi \in \{\mathbf{0.5}, 0.6, ..., 0.9, 0.95\}$					0.036	0.025
RR	$\lambda_R \in \mathbb{R}^+$	$\lambda_R \in \{0.001, 0.005, 0.01, 0.05, \mathbf{0.1}, 0.5, 1\}$			0.050	0.054		

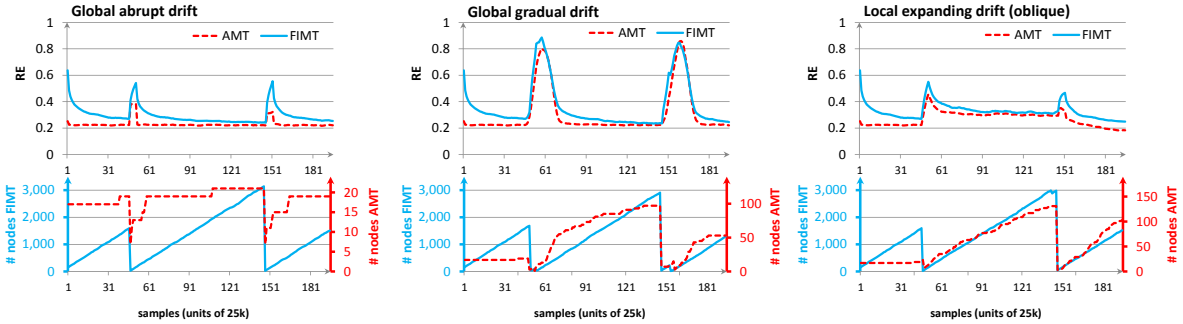

Figure 3. Learning curves on Fried dataset and the corresponding number of tree nodes

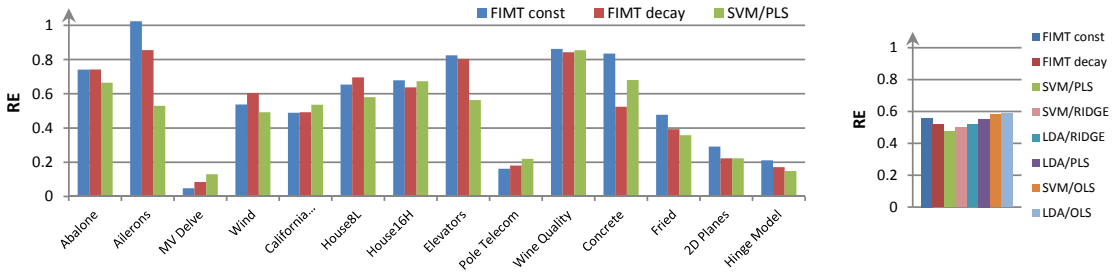

Figure 4. Accuracy comparison: AMT versus FIMT, results of tenfold cross-validation.

AMT and its parameters. The different components of AMT come with parameters. To show that these are robust, we performed several runs in which we kept all parameters constant and varied only one. In Table 1 we provide an overview of the parameters, the values varied, and the resulting standard deviation of RE. The bold parameter values are the ones used for the remaining experiments. The results are averaged over the specified datasets. For the wide value range tried for each parameter, the RE does not vary much.

Time-changing streaming data. Of high interest is the evaluation of AMT on streaming data. We consider the real world Power Consumption dataset and compared the learning curves of AMT and FIMT. Since the memory required for a node is approximatively equal for AMT and FIMT, we evaluate the model size in the number of nodes. For these experiments we used a prequential window of $25,000$ samples, and a sliding step of $5,000$ samples. From the results in Figure 2, we see that FIMT decay has the poorest results, while AMT (with LDA splits) achieved the best accuracy, with a far smaller model.

To investigate the capability of AMT to adjust to the different types of drift (global abrupt drift, global gradual drift, local drift) we use the synthetic dataset Fried and modify it as proposed in [6], creating 10^6 samples. For the local expanding drift we modified the drift regions such that they are oblique instead of axis

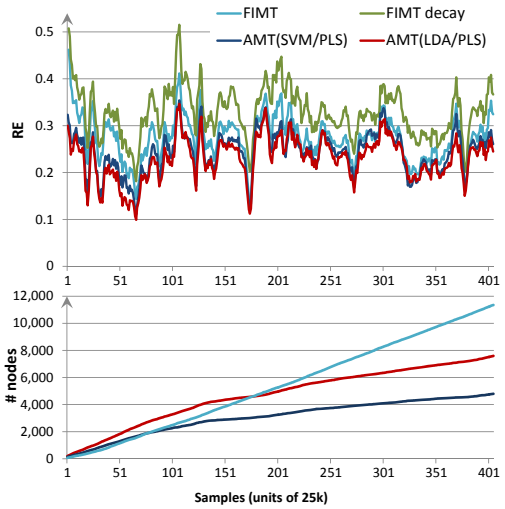

Figure 2. Power Consumption dataset

parallel. For AMT we used LDA splits, since these were more efficient and accurate in our experiments on streaming data, and OLS models, since this data does not contain subspace models. The results are plotted in Figure 3. Here we can clearly see the advantage of the oblique splits, as FIMT builds thousands of nodes, while AMT less than 200. For these experiments, we registered following number of samples per second:

	FIMT	AMT / LDA	AMT / SVM
Global abrupt drift	20,385	**43,676**	13,441
Global gradual drift	22,932	**40,239**	13,550
Local expanding drift	20,083	**41,295**	15,256
Power Consumption	5,281	**7,085**	4,486

We see that AMT is more accurate and more compact than FIMT, and with LDA it is also more efficient.

AMT and FIMT on the artificial and medium data. To allow a comparison with the results presented in [6], we used as evaluation method 10-fold cross-validation and available results from [6]. From the results plotted in Figure 4, we see that in average AMT with SVM as classifier for the split and PLS as model is the most accurate, and almost always more accurate than FIMT. While FIMT processed 2,778 samples per second, AMT (PLS/LDA) processed 5,600 samples per second, and AMT (PLS/SVM) 580 samples per second.

6. Conclusion

We introduced AMT for online prediction of continuous values on streaming data. We proposed different exchangeable components, both for models and split training, and evaluated their accuracy and efficiency. Among the linear models PLS turned out to be the most accurate, as it is able to identify linear subspace models. SVM turned out to be less efficient than LDA, but better suited for high dimensional data. Altogether, we achieved more compact models and accurate results than the state of the art FIMT.

Acknowledgements. The authors gratefully acknowledge the financial support of the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB-686 Model-Based Control of Homogenized Low-Temperature Combustion.

References

[1] C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, pages 70–81, 2000.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[3] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, pages 1579–1619, 2005.

[4] L. Breiman, J. H. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[5] A. Dobra and J. Gehrke. Secret: a scalable linear regression tree algorithm. In *ACM SIGKDD*, pages 481–487, 2002.

[6] E.Ikonomovska, J. Gama, and S. Džeroski. Learning model trees from evolving data streams. *Data Mining Knowledge Discovery*, 23(1):128–168, 2011.

[7] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991.

[8] J. Gama, R. Sebastiao, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *ACM SIGKDD*, pages 329–338, 2009.

[9] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 28(126):505–535, 1974.

[10] A. Lorber, L. E. Wangen, and B. R. Kowalski. A theoretical foundation for the pls algorithm. In *Journal of Chemometrics*, volume 1, pages 19–31, 1987.

[11] R. F. Merino and M. Nunez. Self-adaptive induction of regression trees. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(8):1659–1672, 2011.

[12] D. Potts. Incremental learning of linear model trees. In *ICML*, pages 84–, 2004.

[13] S. J. Qin. Recursive pls algorithms for adaptive data modeling. *Computers and Chemical Engineering*, 23(1):503–514, 1998.

[14] J. R. Quinlan. Learning with continuous classes. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[15] H. Wold. *Estimation of Principal Components and Related Models by Iterative Least squares*, pages 391–420. Academic Press, New York, 1966.