Online Meta-Forest for Regression Data Streams

Ammar Shaker
NEC Laboratories Europe GmbH
Heidelberg, Germany
ammar.shaker@neclab.eu

Christoph Gärtner
BridgingIT GmbH
Mannheim, Germany
christoph.gaertner@bridging-it.de

Xiao He
Alibaba Group
Hangzhou, China
xiao.he@me.com

Shujian Yu
NEC Laboratories Europe GmbH
Heidelberg, Germany
Shujian.Yu@neclab.eu

***Abstract*—Stream learning is essential when there is limited memory, time and computational power. However, existing streaming methods are mostly designed for classification with only a few exceptions for regression problems. Although being fast, the performance of these online regression methods is inadequate due to their dependence on merely linear models. Besides, only a few stream methods are based on meta-learning that aims at facilitating the dynamic choice of the right model. Nevertheless, these approaches are restricted to recommend learners on a window and not on the instance level. In this paper, we present a novel approach, named Online Meta-Forest, that incrementally induces an ensemble of meta-learners that selects the best set of predictors for each test example. Each meta-learner has the ability to find a non-linear mapping of the input space to the set of induced models. We conduct a series of experiments demonstrating that Online Meta-Forest outperforms related methods on 16 out of 25 evaluated benchmark and domain datasets in transportation.**

***Index Terms*—Learning from Data Streams, Adaptive Learning, Meta-Learning, Regression Streams, Data Streams, Online Bagging, Ensemble Learning**

I. Introduction

Classical batch learning considers inducing models from fixed sets of training data, where these models are then deployed in the test environment. This learning paradigm is justified under the assumption of independent and identically distributed *(i.i.d)* samples. In online learning, on the other hand, the data is not available as a whole at any point in time of the learning process, but it is only observed one example at a time.

Learning from data streams poses challenges [2] such as the infeasibility of storing all the data, since the data stream may be infinite. Each instance of the stream can be observed and processed only once and is then discarded. Furthermore, the data-generating processes are subject to change, which is also known as concept drift [23], [17]. These limitations demand highly adaptive systems that fundamentally overcome these challenges where their batch counterparts would fail [11].

Many stream learning methods have been proposed in the last decades to overcome these challenges as described in the survey paper [17]. Most of them (e.g., [27]) focus on classification problems, but only a few can be applied to regression problems [24]. AMRules [1], [13] and FIMTDD [21] are two such examples, which are rule-based and tree-based approaches respectively. They have proven their practicality in terms of runtime and often achieving a good generalization performance. This, however, comes with the limitation of restricting the rules' consequents and leaf nodes to employ linear functions. Therefore, the prediction of both methods are often not satisfactory when the underlying problem are complex non-linear functions. Gomes et al.,[18] propose a random forest of FIMTDD that is learned by applying online bagging on regression streams.

Ever since meta-learning [5] was introduced for data streams in [34], i.e., choosing the right learner from hypothesis spaces of different complexities in the adaptive stetting, algorithm selection has attracted a lot research. *MetaStream* [32] and BLAST [36] are two such examples. Besides being tailored to classification, the former's meta-learning scheme is not suitable for anytime prediction since the algorithm recommendation occurs on a window level. The latter, BLAST, does indeed recommend a learner on instance level using the past performance of the base-learners, as opposed to using the instance's features. As a result, BLAST's performance is upper-bounded by the performance of the best seen model induced so far.

To overcome the aforementioned issues, in this work, we propose a new incremental meta-learning approach, called Online Meta-Forest (OMF), for regression data streams based on the Meta-Decision Tree (MDT) by [33]. It incrementally induces an ensemble of tree-structured meta-learners that recommend the best set of predictors for each test example. To summarize, the contributions of this paper are threefold:

1) We develop a novel incremental meta-learner on regression data streams that recommends at the instance-level. Moreover, the induced meta-learner is able to find a non-linear mapping of the input space to the set of induced models.
2) We provide the guarantees of the online Meta-Tree to recommend the highest-performing model.
3) The experiments demonstrate strong results of Online Meta-Forest on 24 benchmark and real-world datasets.

Over the remainder of this paper, we present the related work in Section II and the details of OMF in Section III, and show empirical evaluation in Section IV before concluding in Section V.

II. Related Work

In the realm of learning from data streams, two main properties for a learner become important: being incremental and adaptive, since they allow for one-pass learning that maintains a good performance even when changes happen.

Hoeffding trees [10] and adaptive Hoeffding trees [3] belong to the pioneering works that adhere to the steaming requirements and achieve good results for classification on data steams; these approaches exploit statistical guarantees provided by the Hoeffding bound [19]. Similarly, AMRules and FIMTDD apply the same guarantees for the variance reduction as a splitting criterion when learning from regression streams.

Apart from inducing single models, ensemble learning aims at inducing a set of several base learners to solve a specific problem. The idea behind most of ensemble methods is to minimize the generalization error by exploiting the bias-variance trade-off, by reducing the variance component while keeping the bias term unchanged (true when all base learners are chosen from the same hypothesis class). Since training the set of learners on the same data would lead to clones of the same model, different ensemble methods focus on how to diversify the ensemble and how to aggregate the decisions taken by the different learners into a single prediction. Hence, the process of ensemble learning can be split into two phases: *base model generation* and *model integration*. Bagging [6] generates a diverse set of learners by applying bootstrapping on the learning process. To this end, each base model is learned on a replica $\mathcal{D}'$ of the training data $\mathcal{D}$ that is sampled with replacement such that $|\mathcal{D}'| = |\mathcal{D}|$. Sampling with replacement does not simply carry over to the streaming setting, because the size of the data stream is not known in advance. Oza and Russell [29] show that sampling with replacement allows each training instance to be selected with probability $1 - (1 - 1/|\mathcal{D}|)^{|\mathcal{D}|}$, which also means that the probability for each replica to contain k copies of an instance is $\binom{|\mathcal{D}|}{k}(\frac{1}{|\mathcal{D}|})^k(1-\frac{1}{|\mathcal{D}|})^{|\mathcal{D}|-k}$. Hence, k tends to follow a Poisson distribution $\text{Pois}(\lambda = 1)$ when $|\mathcal{D}| \to \infty$, i.e., $\mathbf{P}(k) = \frac{1}{e \cdot k!}$. Oza-Bagging exploits this fact in order to apply bagging in the online setting. This is achieved by taking each newly observed instance $(\boldsymbol{x_i}, y_i)$ in the stream $k_i^{(t)} \sim \text{Pois}(\lambda)$ times for each base learner B_i. Adaptive Random Forest (ARF-Reg) [18] proposes an ensemble version of FIMTDD by applying online bagging for the induction of the ensemble's memebers.

As a higher level of learning, meta-learning deals with learning from the learner's performance and behaviour; it has been defined and introduced in different ways in the last two decades [25]. Meta-learning is defined by Brazdil [5] as the type of learning that considers both the declarative and the procedural bias, which are introduced by the hypothesis space the preference relation on the hypothesis space, respectively. In [25], the authors find the common denominators that characterize meta-learning (i) the ability to adapt with more experience, (ii) the exploitation of the data's meta-knowledge and (iii) the consideration of meta-knowledge from multiple domains. Todorovski and Džeroski [33] propose and introduce the meta-decision tree (MDT) as a structure that defines a hierarchical algorithm recommender that selects a learner to be used for an instance reaching a leaf node of the decision tree. MDTs are induced by applying stacking that uses the base learners' confidences as landmarkers and are proposed only for the classification setting. Recently Khiari et al. [22] introduced MetaBags, which applies bagging in order to induce meta-decision trees for regression, with the objective of reducing the error when selecting a learner over the others. In order to make generalizations about the learning process of different data sets, the meta-learning system needs to describe these data sets adequately. These characteristics are called meta-features, and include, for example, statistical and information-theoretic measures of the data. Another method of characterizing the data, which we shall employ here, is landmarking. More on this can be found in [31].

BLAST [36] is a state-of-the-art meta learning approach that incrementally trains and updates a heterogeneous ensemble of base learners while learning on data streams. BLAST is tailored to classification problems where each learner is up- or down-weighted by a fading factor with respect to its error on the most recent examples; this strategy is also known as the *Online Performance Estimation*. For a new observation, the selection strategy activates the base learner that has the best online performance. Prior works of BLAST can be seen in the work [35].

MetaStream [32] is an ensemble meta-learning method that selects the set of best learners from a pool of regression algorithms that are then used as predictors for a freshly incoming batch of examples. This approach splits up the learning into two levels: (i) the base level where regression models are induced from the arriving data, and (ii) the meta level where meta-instances are generated from each batch/window of instances on which a meta-learner is learned. The way in which the learning at the meta level is achieved, in MetaStream, makes it unsuitable for the test-then-train evaluation, which is why we had to exclude it from our performance comparison in Section 3 of the main text.

In this paper, the authors provide a detailed experimental analysis of their method by applying it to data sets from public transportation and the energy sector.

III. Online Meta-Forest

Online Meta-Forest is an online approach that enjoys the strength of both learning paradigms ensemble and meta-learning and puts them into practice for the learning from regression streams. It spawns a forest of a specific type of decision trees, namely a Meta-Decision Tree (MDT) [33], that was invented for meta-learning. An MDT resembles a decision tree with the exception that for each instance reaching a leaf node, that leaf node selects a base learner to be employed for prediction. Each MDT is associated with two sets: (i) the set of meta-feature generators $\mathbb{F}$ that extends the set of features on which the tree builds its splitting criteria and (ii) the set of base learners $\mathbb{B}$ from which leaf nodes employ models for predictions; MDT is formally defined in Definition 2.

In the following, we will first introduce the concept of meta-feature generators and base learners and their online updates. Then we introduce the online induction version of MDT. Finally, we turn the online MDT into a Meta-Forest by applying the ensemble technique bagging.

A. Meta-Feature Generators and Base Learners

Meta-feature generators, $\mathbb{F} = \{F_i | i \in \{1, ..., M\}\}$, play the role of enriching each instance with information that could be (i) *domain specific* such as statistical and information-theoretic measures of each attribute, or (ii) *model-based features*, where each feature is derived from incrementally trained models, see Definition 1. Knowledge about the density around an instance, as a meta-feature, can be, for example, acquired by knowing the number of examples in its neighborhood and the distance to the closest example by querying a k-NN classifier. Upon observing the regression instance $(\boldsymbol{x_t}, y_t) \in \mathcal{X}^d \times \mathbb{R}$, where $\mathcal{X}^d$ is the input space and the output space is $\mathbb{R}$, the set of its meta-features $\{f_1, \ldots, f_M\} = \bigcup_{i=1}^{M} F_i(\boldsymbol{x_t})$ is generated, so that the extended instance takes the form $(\boldsymbol{a_t}, y_t) := (x_{t1}, \ldots, x_{td}, f_1, \ldots, f_M, y_t)$, i.e., the original attributes, the meta-features and the label. Therefore, $\mathbb{F}$ contains supervised learners whose induction's and model's characteristics helps defining a set of meta-features.

Definition 1. *Model-Based Meta-Feature Generator: A meta-feature generator $F_i \in \mathbb{F}$ is the function $F : \mathcal{X}^d \times \mathbb{M} \rightarrow \mathbb{R}$ that assigns to each query instance $\boldsymbol{x}$ the meta-feature f_i, where $\mathbb{M}$ is a hypothesis space. This function takes a learned model $M_i \in \mathbb{M}$ induced either by the batch algorithm A from the batch data $\{\boldsymbol{x}\} \subset \mathcal{X}^d$, or is being induced and incremental updated by an adaptive algorithm on the sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. Hence, $F_i(\boldsymbol{x}, M_i)$ assigns f_i to $\boldsymbol{x}$ by exploiting the characteristic M around (x). For simplicity, we omit M_i in $F_i(\boldsymbol{x}, M_i)$.*

Regression base learners, $\mathbb{B} = \{B_i | i \in \{1, ..., Q\}\}$, are instances of learners taken from different hypothesis classes with different complexities and capacities. In principle, base learners should be diverse experts under different assumptions of the data distribution. In the studied streaming setting, the data distributions could change over time such that certain base learners will perform better at different times; this effect will be perceived and learned by the MDT.

Models from both sets $\mathbb{F}$ and $\mathbb{B}$ are updated in a supervised manner after observing each new training instance $(\boldsymbol{x_t}, y_t)$ only once. Since these updates are performed incrementally, there is no need to explicitly store and maintain the seen examples. This is only possible when each learner can be trained incrementally, otherwise, batch learners can be used by learning on a sliding window as suggested in [8].

B. Online Meta-Decision Tree Induction

Building upon the concept of meta-decision trees, see Definition 2, the induction of the online MDT applies the recursive variance reduction criterion for finding the right feature and splitting threshold. Variance reduction (VR) considers the decrease in variance (in the output) caused by splitting the set of instances Z_p at a leaf node p, into the two sets Z_l and Z_r based on the attribute a_q and one of the values it takes v; i.e., $Z_l = \{(\boldsymbol{a}, y) \in Z_p | a_q \leq v\}$, $Z_r = \{(\boldsymbol{a}, y) \in Z_p | a_q > v\}$. The variance reduction is defined as the difference of the variance at p and the weighted variances after the split:

$$VR_v = Var(Z_p) - \frac{|Z_l|}{|Z_p|} Var(Z_l) - \frac{|Z_r|}{|Z_p|} Var(Z_r). \quad (1)$$

where $Var(Z)$ is the output variance of the set Z, $|Z_p|$, $|Z_l|$, and $|Z_r|$ are the number of samples in node p, its new left and right leaf nodes, respectively. Decision trees and rules for regression, such as FIMTDD and AMRules, select the splitting attribute that maximises (1) on the target attribute.

Definition 2. *Meta-Decision Tree (MDT): $T_i = (R_i, \mathbb{F}_i, \mathbb{B}_i)$ is an MDT, with R_i being the root node, $\mathbb{F}_i$ being the set of meta-feature generators, $\mathbb{B}_i$ being the set of base learners. An internal node takes the form $n_j = (F_{n_j}, t_{n_j}, n_{j_l}, n_{j_r})$, where t_{n_j} is the splitting point used for the meta-feature generated by $F_{n_j} \in \mathbb{F}_i$, n_{j_l} and n_{j_r} are the left and the right child nodes of n_j. Each leaf node p_j takes the form $p_j = (B_{p_j})$, where $B_{p_j} \in \mathbb{B}_i$ is the base learner recommended by p_j for predictions. For a query instance $\boldsymbol{x}$, the prediction is given as $T_i(\boldsymbol{x}) = R_i(\boldsymbol{x})$, where*

$$n_j(\boldsymbol{x}) = \begin{cases} n_{j_l}(\boldsymbol{x}), & F_{n_j}(\boldsymbol{x}) \leq t_{n_j} \text{ and } n_j \text{ is internal node} \\ n_{j_r}(\boldsymbol{x}), & F_{n_j}(\boldsymbol{x}) > t_{n_j} \text{ and } n_j \text{ is internal node} \\ B_{n_j}(\boldsymbol{x}), & n_j \text{ is a leaf node.} \end{cases}$$

s

An exact computation of the variance reduction caused by each of the possible values of each attribute, can be analytically computed in an incremental manner; this, however, requires a linear number of updates in the number of observations for each attribute. Ikonomovska [20] proposes the extended binary search tree (E-BST) as a data structure that enables the approximate incremental computation of the variance reduction for a set of potential splitting points. The cost of updating this structure is logarithmic in the number of candidate splits C, whereas, computing the variance reduction is linear in C and is independent of the number of observed instances. E-BST is based on the method presented in [16] that facilitates an accurate computation of the information gain while inducing Hoeffding trees [10].

Whether the split $a_{ti} \leq v_{iq}$ causes indeed the highest variance reduction can, theoretically, be determined only after observing all the data, which is not an option when learning from data streams. Concentration bounds, such as the Hoeffding bound [19], play a central role in providing statistical evidence of the performance of a parameter without observing the whole population, which has been proven to be effective for adaptive classification and regression problems, see [10]. We employ the Hoeffding inequality to obtain an early evidence that a given split leads to the guaranteed largest variance reduction in comparison to other splits. To this end, we find $VR_{best}^{(t)}$ and $VR_{2best}^{(t)}$, the best and second best achieved variance reductions up to the tth instance. At a given reduction ratio $R^{(t)} = \frac{VR_{2best}^{(t)}}{VR_{best}^{(t)}}$, the split is accepted and

Algorithm 1: UpdateMetaDecisionTree

```
Input: $T$: current meta-tree.
$(\boldsymbol{x_t}, y_t)$: new training example at time $t$.
$k_t$ weight of the current instance.
$\mathbb{F} = \{F_i | i \in \{1..M\}\}$: incremental meta-features generators for $T$.
$\mathbb{B} = \{B_i | i \in \{1..Q\}\}$: incremental base learners for $T$.
/* constants: $\delta$: confidence level,                                  */
/* $\tau$: tie-breaking constant, $\gamma$: complexity constant           */
1  Traverse tree $T$ to leaf $p$ that contains $\boldsymbol{x_t}$
2  $n_p$: number of examples seen by the leaf $p$
   /* compute the meta-features                                          */
3  $\{f_1, \ldots, f_M\} = \bigcup_{i=1}^{M} F_i(\boldsymbol{x_t})$
4  for $a_i \in \{x_{t1}, \ldots, x_{td}, f_1, \ldots, f_M\}$ do
       /* use $a_i$ and $k_t$ to compute and store the candidate
          values in an extended binary search tree (E-BST)             */
       /* for each candidate attribute $a_i$, and each candidate
          value $v_{iq}$, compute the incremental error of each
          base learner for the samples in the current split            */
5  $n_p = n_p + k_t$
6  find $a_{Best}, v_{Best}$ that have the largest variance reduction $VR_{Best}$
7  find $a_{2ndBest}, v_{2ndBest}$ that have the 2nd largest variance reduction
   $VR_{2ndBest}$
8  $\varepsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2t}}$
   /* update the complexity term                                         */
9  $\gamma_{new} = \gamma \cdot \exp(-\frac{1}{|T|})$
10 $\overline{X} = \frac{VR_{2ndbest}}{VR_{best}}$
11 if $((\overline{X} + \varepsilon + \gamma) < 1$ OR $\varepsilon < \tau)$ then
12     replace $p$ by $(p_{left}, p_{right})$ based on the attribute $a_{Best}$ and the
       splitting value $v_{Best}$
13     choose $B_{left}, B_{right}$ that have the lowest errors for the new leaves
       $p_{left}, p_{right}$
14 return $T$
```

performed (at time t), upon observing

$$R^{(t)} + \varepsilon < 1, \quad (2)$$

where $\varepsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2t}}$, $R = 1$ (the width of the ratio's domain [0,1]), and δ is the confidence level. Fulfilling the previous inequality means that the true ratio is less than one, i.e., $\mathrm{E}[R^{(t)}] < 1$, with probability $1 - \delta$, since the Hoeffding inequality tells us that the difference between the observed ratio $R^{(t)}$ and the true ratio $\mathrm{E}[R^{(t)}]$ is less than ε. The aforementioned steps for the meta-tree induction are summarized in Algorithm 1.

In order to avoid the excessive growth of the meta-tree which leads to overfitting, we apply a penalty criterion that controls the number of accepted splits in grown trees. To this end, we add the complexity γ to the left side of the inequality (2) (Line 11, Algorithm 1), where γ is weighted by a factor that is negative exponential with the tree size, i.e., $\gamma_{new} = \gamma \cdot \exp(-\frac{1}{|T|})$ (Line 9).

Once the splitting condition is fulfilled at a given leaf node p, this node is split into two leaf nodes p_{left} and p_{right}. At each new leaf node, we select the base learner with the least generalization error for the instances covered by that leaf, i.e.,

$$B_u = \underset{B \in \mathbb{B}}{\operatorname{argmin}} \sum_{(\boldsymbol{x}, y) \in \mathcal{Z}_u \subset \mathcal{Z}} \ell(y, B(\boldsymbol{x})) \ , \quad (3)$$

where ℓ be a loss function and Z_u is the set of instances reaching the tree node u and $u \in \{p_{left}, p_{right}\}$. It is important to note that the sum over the losses in (3) is also attainable incrementally by a simple modification of the E-BST structure.

In the following proposition, we show how a single meta-tree is eventually recommending, for each query instance, the base learner with the lowest generalization error.

Proposition 1. *Let h_{MT} be an induced meta-decision tree that defines how a set of induced base learners $\mathcal{B} = \{B_j\}_{j=1}^{Q}$ should be aggregated. These base learners are chosen from a set of hypothesis spaces $\mathcal{M} = \{\mathcal{M}_j\}_{j=1}^{Q}$. Let ℓ be a loss function and R_i be the ratio of performances between the best and the second best base learner for a single example $(\boldsymbol{x_i}, y_i)$. Let further P be the number of leaf nodes in h_{MT}. For a given $\delta > 0$, h_{MT} chooses the base learner with the lowest generalization error with probability $1 - \delta$ when for each leaf node $l_p, p = 1, \ldots, P$ it holds that $\bar{R} + \sqrt{\frac{ln(\frac{1}{\delta})}{2N_p}} < 1$, where $\bar{R}$ is the observed mean of R_i and N_p is the number of samples seen at leaf node l_p.*

Proof. We start by recalling the Hoeffding inequality. This concentration bound quantifies the probability of the deviation between the sample mean $\bar{Z}$ of N independent realizations $Z \in [a, b]$ and the expectation $\mathbb{E}[Z]$ to be $P\left(\left|\bar{Z} - \mathbb{E}[Z]\right| \geq \varepsilon\right) \leq 2e^{\frac{-2N \cdot \varepsilon^2}{(b-a)^2}}$.

Let $B_{best}^{(p)}$ and $B_{2best}^{(p)}$ be the two best base learners selected at the leaf node l_p for future prediction after observing N_p samples:

$$B_{best}^{(p)} = \underset{B_j \in \mathcal{B}}{\operatorname{argmin}} \frac{1}{N_p} \sum_{k=1}^{N_p} \ell\left(y_k, B_j(x_k)\right)$$

$$B_{2best}^{(p)} = \underset{B_j \in \mathcal{B} \setminus \{B_{best}^{(p)}\}}{\operatorname{argmin}} \frac{1}{N_p} \sum_{k=1}^{N_p} \ell\left(y_k, B_j(x_k)\right).$$

Further, R_i is given by $R_i = \frac{\ell\left(y_i, B_{best}^{(p)}(\boldsymbol{x_i})\right)}{\ell\left(y_i, B_{2best}^{(p)}(\boldsymbol{x_i})\right)}$. Since the instances $(x_1, y_1), \ldots, (x_{N_p}, y_{N_p})$ observed at the leaf l_p are jointly independent, the ratios $R_1, \ldots, R_{N_p} \in [0, 1]$ are also independent. Therefore, we have $P\left(\left|\mathbb{E}[R] - \bar{R}\right| \geq \varepsilon\right) \leq 2e^{-2N_p \cdot \varepsilon^2}$, which yields $P\left(\mathbb{E}[R] - \bar{R} < \varepsilon\right) > 1 - e^{-2N_p \cdot \varepsilon^2}$, and by setting $\delta = e^{-2N_p \cdot \varepsilon^2}$, we yield that when $\bar{R} + \sqrt{\frac{\ln(\frac{1}{\delta})}{2N_p}} < 1$, we can deduce that $\mathbb{E}[R] < 1$ and, hence, h_{MT} chooses the best prediction with probability $1 - \delta$ at the leaf node l_p.

From this, it can be directly concluded that the meta-tree chooses the best learner with probability $1 - \delta$ as long as every leaf fulfils $\bar{R} + \sqrt{\frac{\ln(\frac{1}{\delta})}{2N_p}} < 1$. $\square$

C. From Online MDT to Online Meta-Forest

Ensemble learning tries to boost the performance of learning methods through exploiting the bias-variance trade-off, by reducing the variance component while keeping the bias term unchanged (true when all base learners are chosen from the same hypothesis class). This has a proven theoretical and empirical effectiveness when applying bootstrap aggregating (Bagging) [6] on decision trees to form Random Forests [7]. In

Algorithm 2: Online Meta-Forest

```
Input: $(\boldsymbol{x_t}, y_t)$: new training example at time $t$.
$\mathbb{T} = \{T_j | j \in \{1..L\}\}$: set of meta-decision trees.
$\mathbb{F}^{(j)} = \{F_i^{(j)} | i \in \{1..M\}\}$: incremental meta-features generators for $T_j$.
$\mathbb{B}^{(j)} = \{B_i^{(j)} | i \in \{1..Q\}\}$: incremental base learners for $T_j$.
1  for $j \in \{1..L\}$ do
2      draw weight $k_t^{(j)} \sim Poisson(1)$
       /* precompute the error estimators per meta-tree */
3      $Z_t^{(j)} = \alpha \cdot Z_{t-1}^{(j)} + k_t^{(j)}$ /* normaliz. factor $Z_t^{(j)}$ */
4      $wSSE_t^{(j)} = \frac{Z_{t-1}^{(j)}}{Z_t^{(j)}}(\alpha \cdot wSSE_{t-1}^{(j)}) + \frac{k_t^{(j)}}{Z_t^{(j)}}(y_t - T_j(\boldsymbol{x_t}))^2$
       /* update the meta-features generators */
5      for $i \in \{1..M\}$ do
6          update $F_i^{(j)}$ on $(\boldsymbol{x_t}, y_t)$, $k_t^{(j)}$ times
       /* update the base learners */
7      for $i \in \{1..Q\}$ do
8          update $B_i^{(j)}$ on $(\boldsymbol{x_t}, y_t)$, $k_t^{(j)}$ times
9      UpdateMetaDecisionTree($T_j$, $(\boldsymbol{x_t}, y_t)$, $k_t^{(j)}$, $\mathbb{F}^{(j)}$, $\mathbb{B}^{(j)}$)
10 return $T$, $\mathbb{F}$, $\mathbb{B}$
```

a stream setting, maintaining bootstrapping poses a challenge due to the inaccessibility of the whole data in advance. This is achieved by incorporating each newly observed instance $(\boldsymbol{x_i}, y_i)$ in the stream $k_i^{(t)} \sim \text{Pois}(\lambda)$ (Poisson distribution) times for each learner in the ensemble, following [29][1].

Our approach extends the online MDT to a forest of meta-trees $\mathbb{T} = \{T_1, ..., T_L\}$ to improve the generalization performance. Each meta-tree T_j updates its structure and parameters from an incrementally constructed bag of examples (replica of the stream). To this end, upon observing the instance $(\boldsymbol{x_t}, y_t)$, it is replicated $k_t^{(j)} \sim \text{Pois}(1)$ times. The copies are used for the incremental learning of the tree's meta-feature generators $\mathbb{F}^{(j)}$ and base learners $\mathbb{B}^{(j)}$. An extended version using the meta-features, of this instance, is formed as $(\boldsymbol{a_t}, y_t) \coloneqq (x_{t1}, \ldots, x_{td}, f_1^{(j)}, \ldots, f_M^{(j)}, y_t)$ which is then used for the induction of the tree T_j.

Due to the dependence of the online MDT's induction process on the Hoeffding bound, the resulting trees are expected to be non-sensitive to the variations between the replicas caused by the online bagging. Hence, we also introduce the ratio parameter r in Online Meta-Forest that governs the fraction of the attributes considered for candidate splits. For simplicity, we hide this ratio r in the Algorithm 2. For the meta tree T_j, its weight after observing the instance $(\boldsymbol{x_t}, y_t)$ is:

$$\begin{aligned} Z_t^{(j)} &= \alpha \cdot Z_{t-1}^{(j)} + k_t^{(j)}, \\ wSSE_t^{(j)} &= \frac{Z_{t-1}^{(j)}}{Z_t^{(j)}}(\alpha \cdot wSSE_{t-1}^{(j)}) + \frac{k_t^{(j)}}{Z_t^{(j)}}(y_t - T_j(\boldsymbol{x_t}))^2 \end{aligned} \tag{4}$$

where $Z_t^{(j)}$ is the normalization factor at time t, $\alpha \in [0, 1]$ is a forgetting factor, and $wSSE$ stands for weighted sum of squared errors.

At prediction time, for a query instance $\boldsymbol{x}$, the prediction is simply given as the weighted sum of all trees, i.e., $\sum_{T_j \in T} wSSE_t^{(j)} \cdot T_j(\boldsymbol{x})$.

[1]They show k follows $\text{Pois}(\lambda = 1)$ when $|\mathcal{D}| \to \infty$, i.e., $\mathbf{P}(k) = \frac{1}{e \cdot k!}$.

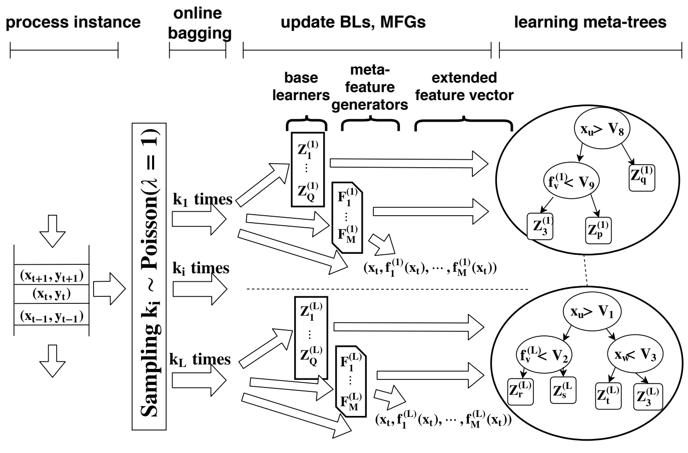

Fig. 1: Procedure of Online Meta-Forest

The functional steps of Online Meta-Forest are shown in Algorithm 2. We illustrate in Figure 1 the general framework of our proposed Online Meta-Forest to let readers have a deep understanding to our methodology. At prediction time, the prediction for a given query instance is given by the convex combination of each recommended base learner's prediction using weights inversely proportional to the weights (4) of the meta-trees.

D. Further Improvements

Instead of evaluating the variance reduction caused by each potential splitting on each instance, one could force this evaluation (starting from Line 6, Algorithm 1) to be carried out in intervals of a predefined length W. A less crude solution that we adopt is to infer a rough estimate of the next evaluation point at which the first condition in Line 11 will be satisfied. To this end, we infer the time for the next evaluation t_{next} from $\varepsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2t_{next}}}$ such that $\varepsilon < 1 - \overline{X} + \gamma$, under the assumption of constant ratio of variance reduction. By accepting t_{next} when $t_{next} \in [t + W/5, t + 5 \cdot W]$, obvious valid splits can be quickly realized, and the evaluation of non-significant splits can be postponed.

Moreover, Hoeffding racing [26] is adapted in order to eliminate candidate splits that show no promising improvements compared to the best two splits. Finally, different components of OMF are subject to drift detection [15] and adaptation.

IV. Experiments

For the purpose of assessing the performance of Online Meta-Forest we conducted experiments using various datasets from different sources, including UCI[2] [12] and other repositories[3]. Additionally, we also used datasets on public transportation of two European capitals $City_1$ and $City_2$; these datasets include scheduling times of buses, route information and the number of passengers boarding and alighting buses at different stops. We use these data to predict two types of demands: (i)

[2]http://archive.ics.uci.edu/ml/

[3]https://github.com/renatopp/arff-datasets/

	Properties			Source and Type		
	#ATT	#INS	RT	ORI	TYP	CP
wine_red	11	1599	[3,8]	UCI	O	R
wine_white	11	4898	[3,9]	UCI	O	R
CASP	9	45730	[15,55]	UCI	O	R
fried_delve	10	40768	[-2,31]	dcc.fc.up.pt	A	R
CCPP	4	9568	[421,496]	UCI	O	R
Concrete	8	1030	[2332,82599]	UCI	O	R
2dplanes	10	40768	[-1000,1000]	dcc.fc.up.pt	O	A
bank32nh	32	8192	[0,0.82	DELVE	O	S
$City_1$**[1]**$\star$	9	191276	[0,184]	–	P	R
$City_1$**[1]**$\bullet$	9	191276	[0,160]	–	P	R
$City_1$**[3]**$\star$	15	175167	[0,147]	–	P	R
$City_1$**[3]**$\bullet$	15	175167	[0,160]	–	P	R
$City_2$**[1]**$\star$	9	46256	[0,96]	–	P	R
$City_2$**[1]**$\bullet$	9	46256	[0,69]	–	P	R
$City_2$**[3]**$\star$	15	46244	[0,96]	–	P	R
$City_2$**[3]**$\bullet$	15	46244	[0,69]	–	P	R
$City_1City_2$**[1]**$\star$	9	237532	[0,184]	–	P	R
$City_1City_2$**[1]**$\bullet$	9	237532	[0,160]	–	P	R
$City_1City_2$**[3]**$\star$	15	221411	[0,147]	–	P	R
$City_1City_2$**[3]**$\bullet$	15	221411	[0,160]	–	P	R
$City_2City_1$**[1]**$\star$	9	237532	[0,184]	–	P	R
$City_2City_1$**[1]**$\bullet$	9	237532	[0,160]	–	P	R
$City_2City_1$**[3]**$\star$	15	221411	[0,147]	–	P	R
$City_2City_1$**[3]**$\bullet$	15	221411	[0,160]	–	P	R
bike_sharing	16	17380	[1,977]	UCI	O	R

TABLE I: Datasets summary. Fields denote the following: #ATT - number of Attributes, #INS - number of Instances, RT - Range of the Target variable, ORI - Origin of data, TYP - Type (**P**roprietary/**O**pen) and CP - Collection Process (**R**eal/**A**rtificial). The symbols $\star$ and $\bullet$ indicate the boarding and alighting datasets, respectively. For the transportation streams, the numbers [1] and [3] represent the lag variable.

boarding: demand of passengers starting from given bus stop, and (ii) alighting: demand of passengers willing to reach a given bus stop. For predicting the demand at a specific time and bus stop, we try out two different settings: (a) offline demand when the trip is planned in the future ($lag = 1$, i.e., no information is available about the demand at the stops previous to this one), and (b) real time demand when more information is available such as the number of passengers at previous stations ($lag > 1$). For the latter demand we use $lag = 3$. In order to evaluate the ability to learn in non-stationary environments, we create an additional stream that simulates a change in the demand prediction task and mimics a transfer learning scenario. To this end, we append the streams of the two cities to impose the change. The characteristics of the used data sets can be seen in Table I.

The question we consider in our evaluations is: How does Online Meta-Forest perform compared to:

1) Base learners used in OMF;
2) State-of-the-art (SoA) meta-learners on data streams (BLAST);
3) State-of-the-art adaptive regression learners (AMRules, FIMTDD and ARF-Reg).

Furthermore, we take advantage of the data stream module from the skmultiflow [28] package which enables us to simulate a streaming context for batch data.

We employ the following learners as base learners of our approach OMF: k-NearestNeighbor (k-NN), HoeffdingTree (HT), GradientBoosting (GRB), RandomForest (RF), GaussianProcesses (GP), Support Vector Regression (SVR), Bayesian Regression (BR); except for HT, all base learners are batch learners and are used as proposed in [8] for combining block-based and online methods. Base learners are used with their default setting[4]. Block-based methods are maintained using a sliding window of size 500 instances and trained after intervals of 100 instances. In the following, we present the meta-feature generators and the features created for a given instance: (i) DecisionTree (DT): the prediction and the depth reached by the instance, (ii) *3*-NearestNeighbor (*3*-NN): the prediction, the distance to the nearest neighbour, and the average distance of all three neighbours (iii) Lasso: the prediction, (iv) StochasticGradienDescent (SGD): the prediction.

As for the parameterization of OMF, we set $r = 0.3$, $W = 200$, and use an ensemble (forest) of size $L = 5$. Since BLAST was only introduced for classification problems, we reformulate the selection criterion by replacing the 0/1 loss with the *mean squared error* (MSE). For an impartial comparison, we allow BLAST to use exactly the same set of heterogeneous base learners that are utilized by OMF.

In order to show the advantage of OMF in comparison to non-meta-approaches, we compare with AMRules, FIMTDD and ARF-Reg that are offered by the stream mining framework MOA[5] [4]. These methods also depend on the Hoeffding inequality, hence, for a fair comparison, we set the same confidence level $\delta = 0.05$ and tie-breaking constant $\tau = 0.005$ for OMF, AMRules and FIMTDD. For ARF-Reg, we use the parameters $\lambda = 6$, the ensemble size $L = 5$, and the number of features $m = \sqrt{d} + 1$ with d being the total number of features, as recommended in the original paper.

In all experiments, we apply the test-then-train procedure, which employs each instance for testing the trained model and then for updating it. The results presented in Table II show the mean over five runs (with different random shuffling of the streams) of the mean squared error (MSE) taken over the whole stream. For an appropriate comparison, we grant all learners a grace period of instances that are ignored for evaluation. This period comprises the first 5% to 10% of examples of the stream. Table II shows how OMF outperforms all learners from all other three categories. (base and meta-learners, and SoA methods). OMF ranks first on 16 out of the 25 datasets with an average rank of 1.66 and has a single tie with BR and BLAST on the 'bank32nh' data.

The last row in Table II shows the number of losses/wins of all methods against OMF; these were computed using the Wilcoxon signed-rank test over the paired performances of the 5 iterations with confidence level $\alpha = 0.05$. It is apparent that OMF statistically wins almost all the time, except against AMRules and BLAST, which lose against OMF in about half of the experiments. OMF loses only twice against HT and BLAST each. Overall, on average, OMF performs best compared to all base learners and SoA methods.

Figure 2 presents a different type of performance evaluation by plotting the MSE on a sliding window; Figures 2.a and

[4]as implemented in scikit-learn [30] and skmultiflow [28].
[5]https://moa.cms.waikato.ac.nz/

	BR	GP	GRB	HT	KNN	RF	SVR	BLAST	AMR	FIMTDD	ARF-Reg	OMF
wine_red	0.44(0.004)	18.18(0.06)	0.43(0.005)	0.54(0.006)	0.62(0.01)	0.44(0.004)	0.55(0.008)	0.43(0.003)	0.48(0.007)	1.03(0.029)	0.48(0.02)	**0.42** (0.005)
wine_white	0.59(0.003)	31.01(0.04)	0.55(0.004)	0.6(0.005)	0.79(0.005)	0.58(0.004)	0.73(0.002)	0.55(0.004)	0.63(0.004)	1.04(0.03)	0.64(0.01)	**0.53** (0.003)
CASP	28.17(0.07)	97.43(0.06)	26.66(0.07)	26.9(0.07)	44.1(0.08)	27.79(0.05)	44.18(0.05)	26.34(0.09)	30.63(1.835)	33.48(0.87)	82.18(51)	**24.36** (0.07)
fried_delve	7.08(0.006)	4.06(0.01)	2.84(0.004)	7.47(0.112)	8.52(0.02)	5.47(0.02)	8.73(0.01)	2.84(0.004)	5.81(0.1)	7.01(0.07)	14.47(2)	**2.78** (0.02)
CCPP	21.1(0.05)	170115(153)	18.7(0.06)	69.1(14.26)	27.1(0.14)	20.5(0.1)	289(0.5)	18.67(0.06)	19.3(0.19)	27(0.56)	33.96(5)	**18.1** (0.12)
Concrete	111.4(1)	1325(7.6)	35.02 (1.03)	236.8(6.5)	107.6(1.2)	43.9(1.4)	276.8(2.5)	**34.89** (1.03)	103.5(5.1)	348.8(9.5)	162.86(17)	35.6(0.62)
2dplanes	5.81(0.006)	3.81(0.003)	1.18(0.001)	1.61(0.02)	2.94(0.005)	1.46(0.002)	2.03(0.002)	1.18(0.001)	1.91(0.04)	2.58(0.05)	13.81(2)	**1.17** (0.002)
bank32nh	**0.0074** (0)	0.021(1e-4)	0.0085(1e-4)	0.0107(1e-4)	0.0091(1e-4)	0.0094(1e-4)	0.011(1e-4)	**0.007** (0)	0.0087(2e-4)	0.022(0.001)	0.015(7e-4)	**0.0074** (1e-4)
$City_1\mathbf{1}\star$	16.95(0.04)	19.5(0.05)	21.73(0.07)	**16.55** (0.05)	20.13(0.05)	21.66(0.05)	18.09(0.04)	16.56(0.05)	16.65(0.05)	16.66(0.05)	16.66(0.07)	16.6(0.04)
$City_1\mathbf{1}\bullet$	20.38(0.05)	23.36(0.05)	25.94(0.05)	**19.93** (0.05)	24.2(0.05)	26.03(0.06)	21.76(0.05)	19.94(0.05)	20.07(0.05)	20.14(0.05)	20.19(0.09)	20.03(0.05)
$City_1\mathbf{3}\star$	13.54(0.03)	17.21(0.04)	16.73(0.02)	13.41(0.04)	15.15(0.04)	16.34(0.04)	15.94(0.04)	13.46(0.02)	13.4(0.13)	13.88(0.07)	24.29(9.35)	**13.12** (0.04)
$City_1\mathbf{3}\bullet$	18.47(0.02)	23.25(0.04)	22.98(0.05)	18.24(0.09)	20.68(0.06)	22.4(0.01)	21.6(0.04)	18.32(0.06)	18.36(0.1)	19.57(0.3)	20.15(0.15)	**17.84** (0.04)
$City_2\mathbf{1}\star$	23.57(0.07)	31331(0.07)	27.87(0.17)	**23.15** (0.04)	27.89(0.1)	29.2(0.12)	26.4(0.09)	23.16(0.04)	23.38(0.1)	23.66(0.18)	24.56(0.2)	23.17(0.08)
$City_2\mathbf{1}\bullet$	21.71(0.06)	29332(0.1)	25.5(0.1)	**21.39** (0.07)	25.73(0.07)	26.9(0.1)	24.21(0.08)	21.3(0.07)	21.38(0.09)	22.73(0.15)	21.3(0.1)	21.31(0.08)
$City_2\mathbf{3}\star$	21.99(0.12)	33.1(0.15)	25.32(0.16)	21.88(0.15)	25.55(0.16)	25.99(0.12)	26.6(0.13)	21.86(0.13)	21.83(0.17)	23.58(0.4)	22.56(0.3)	**21.55** (0.12)
$City_2\mathbf{3}\bullet$	20.32(0.08)	31.0(0.09)	22.61(0.08)	19.98(0.11)	23.62(0.11)	23.67(0.11)	24.15(0.09)	19.98(0.1)	20.15(0.09)	21.67(0.1)	20.27(0.13)	**19.68** (0.09)
$City_1City_2\mathbf{1}\star$	18.28(0.03)	21.87(0.04)	22.97(0.05)	18.1(0.04)	21.68(0.03)	23.15(0.03)	19.76(0.03)	17.96(0.04)	**17.83** (0.07)	17.84(0.1)	17.96(0.24)	17.98(0.04)
$City_1City_2\mathbf{1}\bullet$	20.64(0.03)	24.56(0.03)	25.86(0.03)	20.4(0.03)	24.5(0.04)	26.17(0.04)	22.25(0.03)	**20.29** (0.04)	20.37(0.07)	20.89(0.24)	20.52(0.14)	20.33 (0.03)
$City_1City_2\mathbf{3}\star$	15.35(0.02)	20.62(0.03)	18.55(0.03)	15.55(0.02)	17.39(0.03)	18.41(0.03)	18.23(0.03)	15.29(0.02)	15.26(0.11)	15.87(0.07)	420.4(337)	**15.02** (0.02)
$City_1City_2\mathbf{3}\bullet$	18.89(0.02)	24.95(0.04)	22.92(0.04)	18.89(0.07)	21.33(0.05)	22.68(0.02)	22.17(0.04)	18.78(0.05)	18.83(0.1)	20.11(0.3)	21.26(0.7)	**18.35** (0.02)
$City_2City_1\mathbf{1}\star$	18.06(0.01)	21.51(0.02)	22.77(0.11)	17.91(0.02)	21.43(0.02)	22.96(0.05)	19.5(0.01)	17.83(0.02)	17.87(0.04)	18.43(0.06)	17.96(0.03)	**17.73** (0.01)
$City_2City_1\mathbf{1}\bullet$	20.54(0.02)	24.3(0.03)	25.87(0.08)	20.3(0.03)	24.38(0.02)	26.13(0.05)	22.11(0.02)	20.27(0.03)	20.25(0.05)	20.83(0.08)	20.64(0.1)	**20.18** (0.03)
$City_2City_1\mathbf{3}\star$	15.06(0.02)	20.12(0.03)	18.27(0.03)	15.06(0.02)	17.03(0.02)	18.08(0.03)	17.87(0.03)	15(0.01)	14.67(0.03)	15.52(0.11)	15.62(0.1)	**14.61** (0.01)
$City_2City_1\mathbf{3}\bullet$	18.77(0.03)	24.66(0.01)	22.77(0.05)	18.53(0.02)	21.19(0.02)	22.59(0.06)	22.04(0.01)	18.57(0.02)	**18.1** (0.04)	19.13(0.07)	19.24(0.06)	18.19(0.02)
bike_sharing	16806(28)	11684(24)	7031(15)	17233(230)	13679(72)	8394(35)	32190(54)	7031(15)	14555(318)	15971(483)	19670(200)	**6707.59** (47)
∅ **Rank**	5.88	10.6	7.58	4.94	8.6	8.54	9.04	2.4	3.94	7.18	7.6	**1.7**
Loss/Win	24/0	25/0	21/0	21/2	25/0	25/0	25/0	15/2	13/0	23/0	18/0	N/A

TABLE II: Performance comparison of the algorithms in terms of MSE and the standard error in parentheses. Results with bold font highlight the dataset-wise minimum achieved MSE, and underlined results in the case of ties. The symbols $\star$ and $\bullet$ indicate the boarding and alighting datasets, respectively. For the transportation streams, the numbers [1] and [3] represent the lag variable.

2.b show the progress of MSE while learning from two pure streams of transportation data from the cities Cap_1 and Cap_2, respectively. Figure 2.c compares the MSE of all learners on the interesting problem of concept change where each learner is confronted with data coming from $City_2$ only after learning the concept of $City_1$. From this figure, it is clear that OMF has the smallest error on the whole data stream, even when the change occurs. Furthermore, OMF shows the smallest increase of error at the center of the change after inspecting the first 185000 examples compared to the other methods. Methods that have high error with performance curves above the plotted region are represented with arrows pointing up.

Figure 3 depicts an overall runtime comparison between a single MDT (i.e., OMF with $L = 1$) and BLAST in terms of CPU-seconds against the size of the stream (the same streams used for Table II), where both methods use the exact same base learners. The time needed by our method resembles that by BLAST, despite the overhead that has a sub-linear growth in the stream's length. This time overhead can be explained by the time needed to induce a meta-tree and incrementally evaluate its potential extensions. OMF shows its applicability for high-speed learning on data streams. For instance, on the stream with 221k instances, each instance requires less than 0.278 seconds for prediction and training.

Since statistical tests are often criticized, mainly due to the multiple testing issue, we also compute the critical difference (CD) for all methods including the base learners using the Bonferroni-Dunn test [14] at the confidence level 0.05, as suggested by [9]. Figure 4 shows with a CD of $CD = 3.33$. On the one hand, all stream learning methods perform better than their base learners counterparts. On the other hand, only

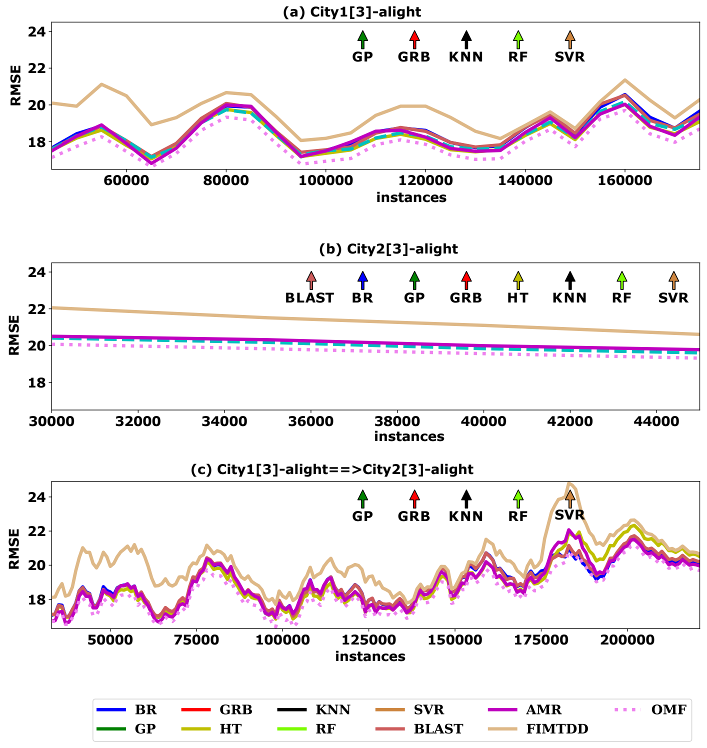

Fig. 2: Performance comparison in terms of the moving mean squared error on three transportation data streams. Methods that have high error with performance curves above the plotted region are represented with arrows pointing up.For the transportation streams, the numbers [1] and [3] represent the lag variable.

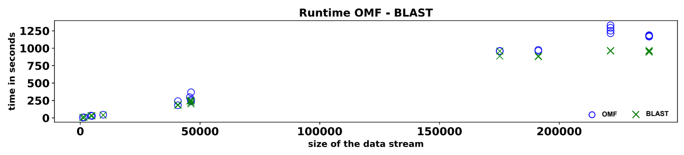

Fig. 3: Performance comparison in terms of the overall runtime (in seconds) between BLAST and Online Meta-Forest (normalized for ensemble). The x-axis depicts the size of the stream.

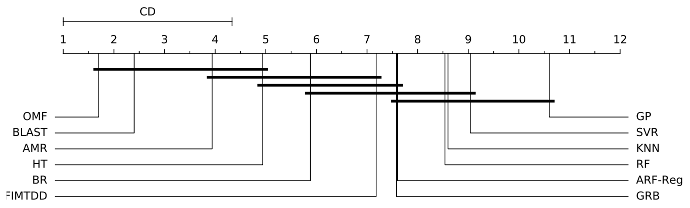

Fig. 4: Comparison of all learners classifiers against each other Bonferroni-Dunn test.

OMF and BLAST are significantly different from benchmark stream learning methods like BR and FIMTDD (as indicated by the second grouping marked with thick solid line).

V. Conclusion

In this paper, we tackle the problem of learning from regression data streams. Our approach incorporates meta-learning on the instance level in order to recommend for each test instance the best set of predictors and their aggregation. To this end, we develop an incremental induction mechanism for meta-decision trees and prove guarantee bounds on their generalization performance. We further show how a single tree can be expanded into the Online Meta-Forest. Our wide-ranging experiments, covering 24 datasets and several competitive algorithms, provide the evidence that OMF performs more accurately than SoA competitors on benchmark datasets and the real-world problem of transportation demand. In the future, we plan to incorporate the dynamic tuning of the hyperparameters of both the base learners and the meat-feature generators into the induction process.

References

[1] E. Almeida, C. A. Ferreira, and J. Gama. Adaptive model rules from data streams. In *ECML PKDD 2013*.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of PODS*, Madison, WI, USA, 2002.

[3] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Proc. of IDA 2009, the 8th Int. Symp. on Intelligent Data Analysis*, pages 249–260, Lyon, France, 2009.

[4] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[5] P. Brazdil, C. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer, 2008.

[6] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[7] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[8] D. Brzezinski and J. Stefanowski. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265, 2014.

[9] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. of SIGKDD*, Boston, MA, USA, 2000.

[11] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

[12] D. Dua and E. Karra Taniskidou. UCI machine learning repository, 2017.

[13] J. Duarte, J. Gama, and A. Bifet. Adaptive model rules from high-speed data streams. *ACM Trans. on Knowledge Discovery from Data*, 10(3):30:1–30:22, 2016.

[14] O. J. Dunn. Multiple comparisons among means. *Journal of the American stat. assoc.*, 56(293):52–64, 1961.

[15] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Proc. of SBIA*, São Luis, Maranhão, Brazil, 2004.

[16] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proc. of SIGKDD*, Washington, DC, USA, 2003.

[17] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comp. Surveys*, 46(4):44:1–44:37, 2014.

[18] H. Gomes, J. P. B., L. Ferreira, and A. Bifet. Adaptive random forests for data stream regression. In *Proc. ESANN*, Bruges, Belgium, 2018.

[19] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[20] E. Ikonomovska. *Algorithms for learning regression trees and ensembles on evolving data streams*. PhD thesis, Jožef Stefan International Postgraduate School, 2012.

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

[22] J. Khiari, L. Moreira-Matias, A. Shaker, B. Zenko, and S. Dzeroski. Metabags: Bagged meta-decision trees for regression.

[23] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. of the 13th Int. Conf. on VLDB*, ON, Canada, 2004.

[24] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

[25] C. Lemke, M. Budka, and B. Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

[26] O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Proc. of NeurIPS*. 1994.

[27] L. L. Minku and X. Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering*, 24(4):619–633, 2011.

[28] J. Montiel, J. Read, A. Bifet, and T. Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5, 2018.

[29] N. C. Oza and S. J. Russell. Online bagging and boosting. In *Proc. 8th Int. Workshop on Artificial Intelligence and Statistics*, FL, USA, 2001.

[30] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] B. Pfahringer, H. Bensusan, and C. G. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *ICML*, 2000.

[32] A. L. D. Rossi, A. C. de Carvalho, C. Soares, and B. F. de Souza. MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, 127:52–64, 2014.

[33] L. Todorovski and S. Dzeroski. Combining classifiers with meta decision trees. *Machine learning*, 50(3):223–249, 2003.

[34] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Algorithm selection on data streams. In *Int. Conf. on Discovery Science*, 2014.

[35] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. Having a blast: Meta-learning and heterogeneous ensembles for data streams. In *2015 IEEE Int. Conf. on Data Mining*. IEEE, nov 2015.

[36] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176, 2018.