A Tree-Adaptation Mechanism for Covariate and Concept Drift

Felipe Leno da Silva [1 2] **Raphael Cobe** [1] **Renato Vicente** [3]

Abstract

Although Machine Learning algorithms are solving tasks of ever-increasing complexity, gathering data and building training sets remains an error prone, costly, and difficult problem. However, reusing knowledge from related previously-solved tasks enables reducing the amount of data required to learn a new task. We here propose a method for reusing a tree-based model learned in a source task with abundant data in a target task with scarce data. We perform an empirical evaluation showing that our method is useful, especially in scenarios where the labels are unavailable in the target task.

1. Introduction and Motivation

Machine Learning (ML) applications are progressively becoming pervasive. Due to recent advances in learning and optimization algorithms, dedicated hardware platforms, and open-sourced implementation frameworks - allied with a huge amount of data being collected and stored in this internet era - ML is being employed in challenging domains of ever-increasing complexity. Learning algorithms try to estimate models that predict $P(Y|X)$, where Y refers to the variable to be predicted (a *class* label in classification problems or a real number for regression), and X refers to the set of relevant features for the prediction problem. Most algorithms assume that the data used for training the ML model (training set) is built by sampling independently from the same joint distribution $P(Y, X)$, with $p(Y, X) = p(Y|X)p(X)$. Although ML is becoming increasingly accessible and most companies are able to build datasets, train, and deploy models for several tasks, learning algorithms are generally data-hungry. Since crafting new training data is still a costly and time-demanding process, additional techniques for reducing the sample-complexity are needed.

[1]Advanced Institute for Artificial Intelligence (AI2), Brazil [2]Lawrence Livermore National Laboratory [3]Serasa Experian. Correspondence to: Felipe Leno da Silva <leno@llnl.gov>.

Proceedings of the LatinX in AI (LXAI) Research workshop at ICML 2021.

Transfer Learning (Silva & Costa, 2019) enables reusing knowledge for reducing sample complexity of learning a new task. However, reusing knowledge from one (or more) task(s) for learning a new one is challenging. The main reason for that is that we can generally expect that the distribution $P(Y|X)$ will change from one (source) task to another (target) task, - which we call here as *Concept Drift* (Webb et al., 2016) - making it hard to directly reuse samples or models. Additionally, bias in the sampling process might cause the distribution $P(X)$ to shift from the training phase to the deployment phase - which we call here as *Covariate Drift*. Therefore, both $P(Y|X)$ and $P(X)$ might change from one task to another.

Despite the above-mentioned challenges, reusing knowledge is useful in many practical situations. We here deal with reusing models across different tasks. Our approach trains a tree model in a task and refines it to another, achieving better performance without fully retraining for the new task.

2. Background

A particular learning task consist of trying to build a predictive model $h : X \rightarrow Y$, where $X = X^1, \ldots, X^f, \forall_{x \in X} x \in \mathbb{R}$ corresponds to a set of features hopefully sufficient for fully describing the task, and Y is the *target* variable to be predicted. $Y \in \mathbb{R}$ in regression tasks and $Y \in C$ in classification tasks, where C is a set of possible labels. A *concept* is defined as the joint distribution: $Concept = P(Y, X)$ (Gama et al., 2014), which can be decomposed as $P(Y, X) = P(Y|X)P(X)$. We are primarily interested in modelling $P(Y|X)$ to predict Y for new samples. However, since $P(Y, X)$, $P(Y|X)$ and $P(X)$ are all initially unknown, we train learning algorithms by gathering a *dataset* of samples with known answers $\boldsymbol{O} = o^1, \ldots, o^n, o^i =< x^i, y^i >$. The learning algorithm then has to learn how to generalize a function to predict Y for new samples based on $\boldsymbol{O}$. Many learning algorithm have been proposed for learning in this scenario. In this paper we focus on learning algorithm derived from *Decision Trees*, such as Tree Gradient Boosting and Random Forests.

Decision Trees are binary tree-like models, where each node corresponds to a *test* specifying the path a particular sample has to follow in the tree. The tree leaves specify the

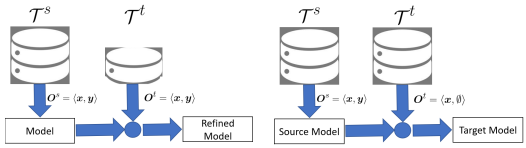

Figure 1. Illustration of the transfer scenarios explored in this paper. The *model refining* scenario (left) and the *model adapting* scenario (right) mainly differ by the availability of samples in the target task.

final output of the algorithm. Usually, each node contains a single test, e.g., verifying if the value of a feature is higher than a threshold. The learning process consists of defining the tree topology and tests (i.e., which feature to test and the corresponding test threshold) for all nodes. Decision Trees are easy to visualize and interpret, making it a popular model. The Tree Gradient Boosting (Chen & Guestrin, 2016) algorithm trains a model in an additive manner and has been shown to be very efficient in real-world tasks. Although those algorithms were able to solve many real-world problems, they require ample access to training data, which might be unrealistic for several tasks. One way of improving data-efficiency is by performing *Transfer Learning*. Formally, we define a *domain* $D = \langle X, Y \rangle$ as a feature space X and an output space Y. A *task* $\mathcal{T} = \langle P(X), f(.) \rangle$ consists of a marginal probability distribution $P(X)$ and an objective predictive function $f(.)$, which is not observable but is the "ground truth" function that generates the correct output for any sample. Tasks belonging to a single domain are expected to be similar, but not equal. The most common way of performing transfer is by reusing knowledge from a source task $\mathcal{T}^s$ to a target task $\mathcal{T}^t$. In case $P^s(X) \neq P^t(X)$, we say that a *Covariate Drift* happened from one task to another. In case $f^s(.) \neq f^t(.)$, we say that a *Concept Drift* happened. Knowledge can be reused in several ways, such as reusing $\boldsymbol{O}^s$ as additional samples for learning $\mathcal{T}^t$ (*instance-transfer*). We focus here on reusing and refining $h^s(.)$ in the target task (*model-transfer*). Performing transfer is useful but generally hard because of covariate and concept drift.

3. Problem Statement

We consider two different transfer scenarios illustrated in Figure 1. For both scenarios, we are interested in gathering knowledge from a source task $\mathcal{T}^s$ and reusing it in a target task $\mathcal{T}^t$. They both belong to a same domain[1] D, $\mathcal{T}^s \in D$ and $\mathcal{T}^t \in D$. For both scenarios, we are only concerned with the model performance in $\mathcal{T}^t$, and the source task is considered only as a mean to improve performance in $\mathcal{T}^t$.

[1]We are not interested in transfer scenarios where the feature and/or output space differ across tasks.

In the *Model Refining* scenario, a labeled training set is available for each of the tasks ($\boldsymbol{O}^s$ and $\boldsymbol{O}^t$). If the transfer procedure is efficient, $l(\mathcal{T}^t, h(\boldsymbol{O}^s \cup \boldsymbol{O}^t)) < l(\mathcal{T}^t, h(\boldsymbol{O}^t))$, i.e., the loss (inverse of performance) in the target task is lower when leveraging data from $\mathcal{T}^s$ than when using only the data available from $\mathcal{T}^t$. Usually, this scenario makes sense when data from the source task is more abundant than in the target task: $|\boldsymbol{O}^s| > |\boldsymbol{O}^t|$.

In the *Model Adapting* scenario, a labeled training set is only available for $\mathcal{T}^s$. For the target task only unlabeled samples are available: $\forall o \in \boldsymbol{O}^t : o = \langle \boldsymbol{x}, \emptyset \rangle$, which means that a model cannot be directly trained with data from the target task. However, since $\boldsymbol{O}^s$ has labels, this scenario tries to reuse knowledge from the source task while accounting for possible drifts across tasks.

For an example of where those scenarios would be applicable in practice, consider a credit score company. If the company is planning on moving to another country, the labels (which clients defaulted their loans) will be unavailable for a long period of time. However, customer profiles (features) will be available almost immediately, corresponding to a *Model Adapting* transfer scenario, because the company has abundant data on its country of origin. After some time the company will have some labels in the new country, but not enough to train a high-performance model. Then, this corresponds to a *Model Refining* scenario. Therefore, algorithms that deal with either (or both) of the scenarios have practical use in many real-world situations.

4. A Tree-Adaptation Mechanism for Covariate and Concept Drift

We propose a transfer mechanism that can be used for both scenarios described in Section 3. Our method is specialized to tree-based learning algorithms, and fully described in the Algorithm in the supplementary material. Each node v on the decision tree splits the training set $\boldsymbol{O}$ in two subsets $\boldsymbol{O}^l$ and $\boldsymbol{O}^r$ according to their value of a single particular feature. Each split induces a new distribution of labels in each subset, hopefully resulting in leaves with minimal entropy. The transfer mechanism consists of reusing the tree topology from one task to another. However, notice that a drift in $P(X)$ or $P(Y|X)$ will affect the resulting split after each node, reducing the accuracy of the tree model because subsequent nodes will receive data of an unexpected distribution. Therefore, our procedure recalculates all thresholds in the tree, trying to reconstruct the same distributions observed in the source task. For the *Model Refining* scenario we reconstruct the label distributions after each split, while in the *Model Adapting* scenario we reconstruct the distributions $P(x_v)$. Since our high-level motivation is very similar to the *STRUTS* transfer algorithm (Segev et al., 2017), we opted to keep the STRUTS nomination for this algorithm, recog-

nizing that Segev's and our methods belong to a common family of algorithms. However, notice that the calculations to be described next are different from Segev's paper.

We always start by fitting a decision tree to the source task by using its training set $\boldsymbol{O}^s$. Starting from the root node, the algorithm defines the sample subsets $\boldsymbol{O}^s_l$ and $\boldsymbol{O}^s_r$ that were assigned to the left and right child nodes in the source task. Then, a set of new threshold candidates T is generated. Each threshold $\tau^c \in \mathrm{T}$ is then evaluated according to a threshold measure, and the best threshold τ^t is defined. The node v is then updated by setting the new threshold τ^t and the induced splits $\boldsymbol{O}^t_l$ and $\boldsymbol{O}^t_r$ are defined and used the update the left and right subtrees.

For defining the threshold candidates and quality, let $\delta(\boldsymbol{O})$ be a distribution of values assumed by samples in $\boldsymbol{O}$ (we specify how to calculate this distribution for each scenario in the next subsections). The quality of each threshold τ^c is estimated by the *divergence gain* observed by using τ^c. The motivation behind the divergence gain is to select the threshold that induces in the target task a distribution as similar as possible as the one originally observed in the source task for a particular node: $t\_q = 1 - \frac{|\boldsymbol{O}^t_l|}{|\boldsymbol{O}^t_v|} JS(\delta(\boldsymbol{O}^s_l), \delta(\boldsymbol{O}^t_l)) - \frac{|\boldsymbol{O}^t_r|}{|\boldsymbol{O}^t_v|} JS(\delta(\boldsymbol{O}^s_r), \delta(\boldsymbol{O}^t_r))$ where JS is the Jensen-Shannon divergence (which is limited to the range $[0, 1]$), $|\boldsymbol{O}^t_l|$ is the number of samples in the target task assigned to the left subtree, and $\boldsymbol{O}^t_v = \boldsymbol{O}^t_l \cup \boldsymbol{O}^t_r$.

For defining thresholds candidates, every node v in the tree performs a test on feature x_v dividing the source task data in two. The first subset contains the first p-th percentile of the data sorted by x_v, $p = percentile(\boldsymbol{O}^s, x_v, \tau)$, while the second contains the remaining samples. Due to possible drifts, the same threshold will correspond to a different percentile in the target task. Therefore, we generate the candidate thresholds $around$ the threshold τ^t that approximates the percentile p in the target task: $\tau^t = arg\min_{\tau^c} |p - percentile(\boldsymbol{O}^t, x_v, \tau^c)|, thres\_candidates = [\tau^t - \epsilon, \tau^t + \epsilon]$, where ϵ is a parameter specifying how much the threshold candidates are allowed to diverge from τ^t.

The calculation δ is specific for each of the scenarios and is described in the next subsections.

4.1. Model Refining Transfer

In the *Model Refining* scenario, the labels are available for all samples. Therefore, we rely on the label distribution for calculating δ:

$$\delta_{MR}(\boldsymbol{O}) = P(\boldsymbol{O}.y) \tag{1}$$

where we denote as $P(\boldsymbol{O}.y)$ the distribution of labels, discretized as the observed histogram with the number of labels for each class. Integrating δ_{MR} into the method described in the last section results in choosing the threshold that induces similar proportions of classes in the subtrees across tasks. We call our algorithm equipped with δ_{MR} as $STRUTS_{MR}$.

4.2. Model Adapting Transfer

In the *Model Adapting* scenario, labels are unavailable for the target task. Hence, calculating $\delta_{MR}(\boldsymbol{O}^t)$ would be impossible. For this scenario we hence consider the distribution of values for the feature chosen for the split in that node:

$$\delta_{MA}(\boldsymbol{O}) = P(\boldsymbol{O}.x_v) \tag{2}$$

where $P(\boldsymbol{O}.x_v)$ the distribution of values observed in the feature x_v for samples in $\boldsymbol{O}$. We call our algorithm equipped with δ_{MA} as $STRUTS_{MA}$.

5. Experiments

In order to evaluate the effectiveness of our method, we evaluate our algorithm two domains. The first one, namely *circle*, is a toy domain where we can easily control *covariate* and *concept* drift. The other domain, *mushroom*, is a real-world problem where we expect our method could be useful. In this domain, the algorithm tries to define if a mushroom is edible or poisonous based on its morphological features. In the next subsections we describe each domain and the experimental results. In all domains, we evaluate the following algorithms: (i): **OnlySource**: This algorithm trains a model using exclusively $\boldsymbol{O}^s$, completely ignoring any kind of drift that might have happened; (ii): **OnlyTarget**: This algorithm trains a model using exclusively $\boldsymbol{O}^t$ when possible (i.e., in *model refining* scenarios), which means that *OnlyTarget* corresponds to normal Machine Learning without transfer; (iii): $STRUTS_{MR}$ **and** $STRUTS_{MA}$: For *model refining* scenarios, we use $STRUTS_{MR}$ (Section 4.1). For *model adapting* scenarios, we use $STRUTS_{MA}$ (Section 4.2). We consider Gradient Boosting (GB) as the base learning algorithm.

5.1. Circle Domain

Our artificially-generated target task consists of classifying samples as "inside" or "outside" a circle of arbitrary radius r and origin (x_o, y_o).With r, x_o, and y_o unknown, the classifier has to learn to label a new sample (x,y) based on a training set. In the *model refining* scenario, we simulate covariate drift by sampling points in the left side of the circle with a higher probability for $\mathcal{T}^s$, In the *model adapting* scenario, the samples in $\mathcal{T}^s$ are generated by a circle with a different radius, and samples for $\mathcal{T}^t$ have no label. The results in Table 1 refer to the average accuracy observed in 200 repetitions of the experiment.

In the *Model Refining* scenario, *OnlySource* achieves the

best performance. This indicates that the drift in $P(X)$ represents a smaller obstacle to tree models than the scarceness of data. The model trained from abundant but drifted data (*OnlySource*) outperforms the models that make use of the scarce data without drift (*OnlyTarget* and $STRUTS_{MR}$). However, for the *Model Adapting* scenario, samples in the target task cannot be directly used because they have unknown labels. For that reason, we evaluate only *OnlySource* and $STRUTS_{MA}$ in this scenario. Differently from the *Model Refining* scenario, the drift in $P(Y|X)$ severely hampered the model ability to directly reuse the data from the source task. In this scenario, $STRUTS_{MA}$ is the top performer for both algorithms.

Finally, when drift in both $P(X)$ and $P(Y|X)$ are present (our *Model Refining and Model Adapting* scenario), the results are very similar from the *Model Adapting scenario*. The overall accuracy of the algorithms is slightly decreased, but $STRUTS_{MA}$ is still the top performer. In conclusion, in the *circle* domain, whenever a *concept drift* happens it is useful to perform model transfer using $STRUTS$. No benefit was observed when only *covariate drift* happens.

	circle	**mushroom**
Model Refining		
OnlySource (GB)	**86.31**	36.60
OnlyTarget (GB)	81.21	66.08
$STRUTS_{MR}$ (GB)	82.74	**75.97**
Model Adapting		
OnlySource (GB)	57.55	36.60
$STRUTS_{MA}$ (GB)	**60.20**	**58.80**
Model Refining and Model Adapting		
OnlySource (GB)	56.80	N/A
$STRUTS_{MA}$ (GB)	**58.56**	N/A

Table 1. Average accuracy observed in 200 repetitions of our experiments in both domains.

5.2. Mushroom Domain

The *Mushroom* domain is a real-world inspired benchmark that has been widely used for evaluating classification algorithms. The dataset is freely available in the UCI repository[2] and consists of 8124 instances described by 22 categorical features. In this task, the algorithm has to classify whether if a specific mushroom species is *edible* or *poisonous* by evaluating its physical characteristics such as odor and stalk.

In order to simulate concept and covariate drift, we define as *source task* classifying all samples with *enlarging* stalk shape. The *target task* classifies the reaming samples, with *tapering* stalk shape. This sample split is similar as the one carried out for other works studying drifts (Segev et al., 2017). Since we cannot control the distribution $P(Y|X)$ in this domain, we only consider the *model refining* scenario where covariate drift is induced and the *model adapting* scenario with unknown target labels, but drift only in $P(X)$. All results are shown in Table 1. The results from the *model refining* scenario are different from our previous evaluation domain. *OnlySource* has a low accuracy in this domain, which shows that the model was not able to generalize rules for the target task from the source. In the *model adapting* scenario, $STRUTS_{MA}$ remains the top performer, improving in over 20% the accuracy of *OnlySource*.

[2]https://archive.ics.uci.edu/ml/datasets/Mushroom

6. Conclusion

We here proposed a model transfer method specialized to tree-learning base algorithms to reuse knowledge from a source task (usually with abundant data) to a target task (usually with scarce data). We describe the *covariate* and *concept* drift problems, and show variations of our proposed methods specialized for each of them. We evaluate our method in two domains, where one of them is a real-world problem, and one is a toy problem where we can easily control the drift. Our experiments show that the transfer method is useful, especially when concept drift happens and labels are not available in the samples from the target task.

Acknowledgments

We acknowledge partial funding from FUNDUNESP (3061/2019-CCP). Lawrence Livermore National Security, LLC, release number: LLNL-CONF-823596.

References

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Gama, J. a., Žliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), March 2014. ISSN 0360-0300. doi: 10.1145/2523813.

Segev, N., Harel, M., Mannor, S., Crammer, K., and El-Yaniv, R. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE Trans. Pat. Anal. and Mac. Int.*, 39(9):1811–1824, 2017.

Silva, F. L. D. and Costa, A. H. R. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

Detailed Algorithm

Our method is illustrated in a high level in Figure 2. Each node v on the decision tree splits the training set $\boldsymbol{O}$ in two subsets $\boldsymbol{O}^l$ and $\boldsymbol{O}^r$ according to their value of a single particular feature. Each split induces a new distribution of labels in each subset, hopefully resulting in leaves with minimal entropy. The transfer mechanism consists of reusing the tree topology from one task to another. Our procedure recalculates all thresholds in the tree, trying to reconstruct the same distributions observed in the source task when the task changes.

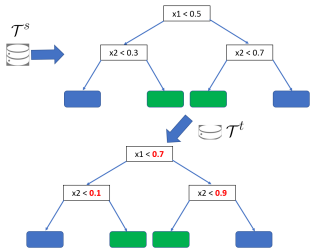

Figure 2. High-level illustration of our method. A tree model is fitted on the training set from $\mathcal{T}^s$. Then, the tree topology is reused for $\mathcal{T}^t$. However, since covariate and/or concept drift is expected to happen, all node thresholds are recalculated based on $\boldsymbol{O}^t$.

Algorithm 1 fully describes the procedure in the high-level. The algorithm is initiated after a decision tree model is fitted to the source task by using its training set $\boldsymbol{O}^s$. The first v is the root node, and each node is composed of a feature x_v and a threshold τ for performing the test, and pointers to the left and right child nodes (v_l and v_r) according to the tree topology.

Starting from the root node, the algorithm defines the sample subsets $\boldsymbol{O}^s_l$ and $\boldsymbol{O}^s_r$ that were assigned to the left and right child nodes in the source task. Then, a set of new threshold candidates T is generated. Each threshold $\tau^c \in$ T is then evaluated according to a threshold measure, and the best threshold τ^t is defined. The node v is then updated by setting the new threshold τ^t and the induced splits $\boldsymbol{O}^t_l$ and $\boldsymbol{O}^t_r$ are defined and used the update the left and right subtrees.

For defining the threshold candidates and quality, let $\delta(\boldsymbol{O})$ be a distribution of values assumed by samples in $\boldsymbol{O}$ (the specific calculation for this is in Sections 4.2 and 4.1). The quality of each threshold τ^c is estimated by the *divergence gain* observed by using τ^c. The motivation behind the divergence gain is to select the threshold that induces in the target task a distribution as similar as possible as the one

Input: node $v = \langle x_v, v_l, v_r, \tau \rangle$, training sets $\boldsymbol{O^t}$ and $\boldsymbol{O^s}$
if $|\boldsymbol{O^t}| > 0$ **then**
  $\boldsymbol{O}^s_l \leftarrow \boldsymbol{O}^s.x_v \leq \tau$
  $\boldsymbol{O}^s_r \leftarrow \boldsymbol{O}^s.x_v > \tau$
  T $\leftarrow$ $thres\_candidates(\tau, x_v, \boldsymbol{O}^s_l, \boldsymbol{O}^s_r, \boldsymbol{O}^s, \boldsymbol{O}^t)$
  $\tau^t \leftarrow$ $arg\max_{\tau^c \in \mathrm{T}} t\_q(\tau^c, x_v, \boldsymbol{O}^s_l, \boldsymbol{O}^s_r, \boldsymbol{O}^s, \boldsymbol{O}^t)$
  $v.\tau \leftarrow \tau^t$
  $\boldsymbol{O}^t_l \leftarrow \boldsymbol{O}^t.x_v \leq \tau^t$
  $\boldsymbol{O}^t_r \leftarrow \boldsymbol{O}^t.x_v > \tau^t$
  $v.v_l \leftarrow STRUTS(v_l, \boldsymbol{O}^s_l, \boldsymbol{O}^t_l)$
  $v.v_r \leftarrow STRUTS(v_r, \boldsymbol{O}^s_r, \boldsymbol{O}^t_r)$
end
return v

Algorithm 1: STRUTS

originally observed in the source task for a particular node:

$$t\_q = 1 - \frac{|\boldsymbol{O}^t_l|}{|\boldsymbol{O}^t_v|} JS(\delta(\boldsymbol{O}^s_l), \delta(\boldsymbol{O}^t_l)) - \frac{|\boldsymbol{O}^t_r|}{|\boldsymbol{O}^t_v|} JS(\delta(\boldsymbol{O}^s_r), \delta(\boldsymbol{O}^t_r)) \tag{3}$$

where JS is the Jensen-Shannon divergence (which is limited to the range $[0, 1]$), $|\boldsymbol{O}^t_l|$ is the number of samples in the target task assigned to the left subtree, and $\boldsymbol{O}^t_v = \boldsymbol{O}^t_l \cup \boldsymbol{O}^t_r$.

The motivation behind our approach for defining threshold candidates is illustrated in Figure 3. Every node v in the tree performs a test on feature x_v dividing the source task data in two. The first subset contains the first p-th percentile of the data sorted by x_v, $p = percentile(\boldsymbol{O}^s, x_v, \tau)$, while the second contains the remaining samples. Due to possible drifts, the same threshold will correspond to a different percentile in the target task. Therefore, we generate the candidate thresholds *around* the threshold τ^t that approximates the percentile p in the target task:

$$\tau^t = arg\min_{\tau^c} |p - percentile(\boldsymbol{O}^t, x_v, \tau^c)| \tag{4}$$

$$thres\_candidates = [\tau^t - \epsilon, \tau^t + \epsilon], \tag{5}$$

where ϵ is a parameter specifying how much the threshold candidates are allowed to diverge from τ^t.

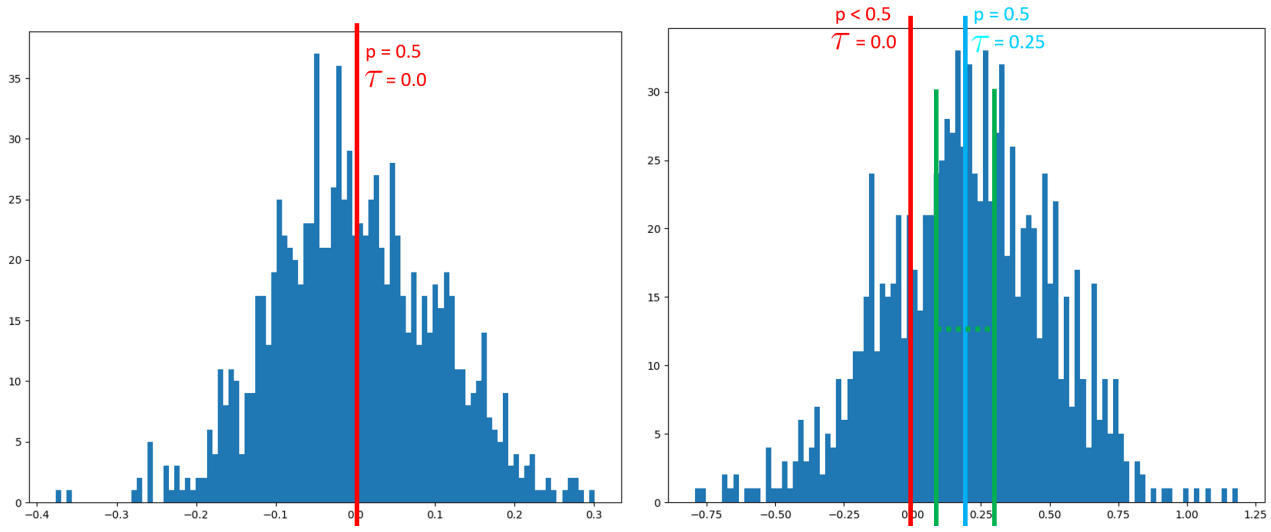

Figure 3. Illustration of the process for generating threshold candidates. Each node in the tree learned from $\boldsymbol{O}^s$ has a threshold τ splitting the data in a percentile p according to the value in feature x_v (a). Due to drift, the same threshold potentially results in a different percentile when splitting $\boldsymbol{O}^t$ (b - red line). We define the new threshold resulting in a split in percentile p (cyan line), and consider a range of values near to that threshold (green lines).