

# Gradient Boosting Algorithms

---

Helder Vieira, up201503395  
Samuel Aduroja, up202009191  
Vânia Guimarães, up200505287

# Gradient Boosting – main ideas

- Ensemble model that combines many weak learners to produce a powerful learner in an additive way.
- Weak learner or base learner is usually a small tree. But we can define another learner.
- Gradient Boost builds predefined  $M$  number of base learners or until additional weak learner fail to improve the fit.
- Each new base learner tries to predict the pseudo-residuals, fixing the previous model, following the direction of negative gradient of loss function.

# Gradient Boosting

**Input:** Training data  $\{(x_i, y_i)\}_{i=1}^n$ , a differentiable **Loss Function**  $L(y, F(x))$ , number of iterations  $M$ .

| Setting               | Loss Function   | $-\partial L(y_i, f(x_i))/\partial f(x_i)$   |
|-----------------------|---|--|
| Regression            | $\frac{1}{2}[y_i - f(x_i)]^2$                             | $y_i - f(x_i)$                               |
| Binary Classification | $-\sum_{i=1}^n (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$ | $y_i - p(x_i)$                               |
| Multi-Classification  | $-\sum_{i=1}^n \sum_{j=1}^K y_i \log(p_i)$                | $K\text{th class: } I(y_i = C_k) - p_k(x_i)$ |

## Step 1)

Initialize model with a constant value:

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$

- $y_i$  is the observed target
- $\gamma$  is our initial guess, the predicted value
- Goal: find a predicted value that minimizes the loss function
- For regression,  $F_0(x)$  is the **average of the observed values**
- For binary classification,  $F_0(x)$  is the **predicted log(odds)**

| Price of oil | Season | Working_Day | Weather situation | Count of train's users | Initial Value $F_0(x)$ |
|--------------|--------|-------------|-------------------|------------------------|------------------------|
| 1,30 €       | Spring | Yes         | Few Clouds        | 1,000                  | 1,372                  |
| 2,00 €       | Winter | Yes         | Snow              | 1,758                  | 1,372                  |
| 1,75 €       | Summer | No          | Clear             | 1,373                  | 1,372                  |
| 1,27 €       | Fall   | No          | Heavy Rain        | 1,469                  | 1,372                  |
| 1, 14 €      | Spring | Yes         | Clear             | 974                    | 1,372                  |
| 1, 58 €      | Fall   | No          | Fog               | 1,659                  | 1,372                  |

## Step 2)

For  $m = 1$  to  $M$ :

1. Compute so-called *pseudo-residuals*:

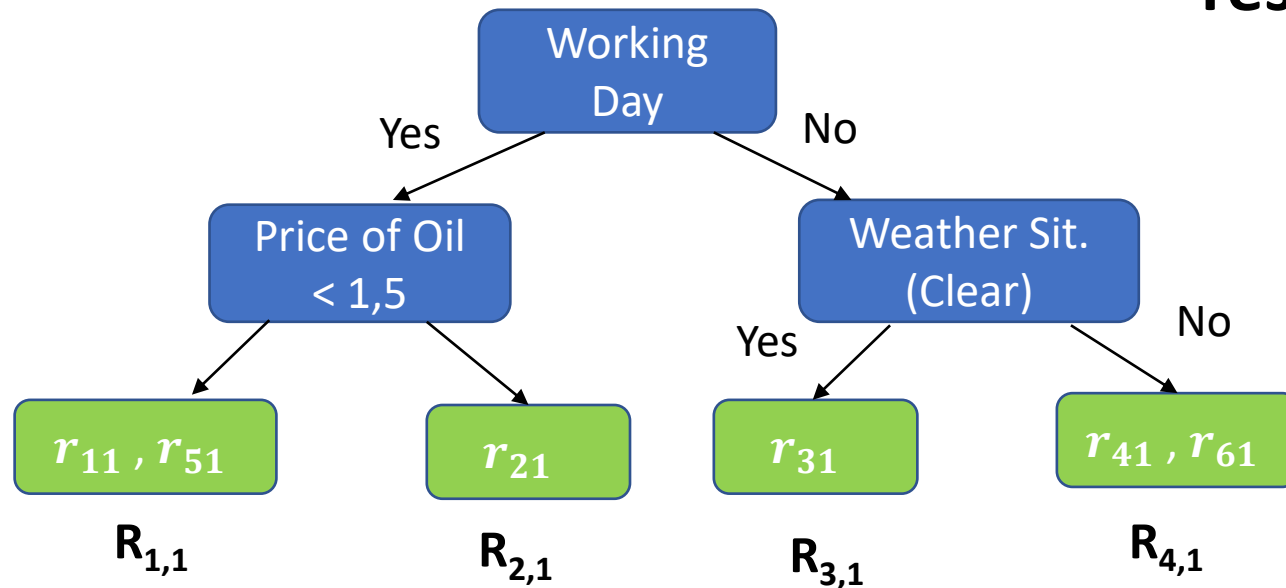
$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

| Price of oil | Season | Working Day | Weather situation | Count of train's users | Pseudo-residuals<br>$r_{i1}^{(1)}$<br>$(y_i - F_0(x))$ |
|--------------|--------|-------------|-------------------|------------------------|--|
| 1,30 €       | Spring | Yes         | Few Clouds        | 1,000                  | -372   |
| 2,00 €       | Winter | Yes         | Snow              | 1,758                  | 386  |
| 1,75 €       | Summer | No          | Clear             | 1,373                  | 1  |
| 1,27 €       | Fall   | No          | Heavy Rain        | 1,469                  | 97   |
| 1, 14 €      | Spring | Yes         | Clear             | 974                    | -398   |
| 1, 58 €      | Fall   | No          | Fog               | 1,659                  | 287  |

**(1) Note:** They are standard residuals with the Squared Error Loss.  
If not, it will be the respective **negative gradient** of the loss function.

**Step 2.2)** Fit a regression tree to the  $r_{im}$  values and create terminal regions  $R_{jm}$ , for  $j = 1, \dots, J_m$

Gradient Boost will build a regression tree to predict the **residuals** instead of the **number of train's users**.



### Step 2.3)

For  $j = 1, 2, \dots, J_m$  compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

- The output value for each leaf,  $\gamma_{jm}$ , is the value that **minimizes this Loss Function**

$$\gamma_{1,1} = \arg \min_{\gamma} \left[ \frac{1}{2} (y_1 - (F_0(x_1) + \gamma))^2 + \frac{1}{2} (y_5 - (F_0(x_5) + \gamma))^2 \right]$$

$$\gamma_{1,1} = \arg \min_{\gamma} \left[ \frac{1}{2} (1,000 - (1,372 + \gamma))^2 + \frac{1}{2} (974 - (1,372 + \gamma))^2 \right]$$

$$\frac{d}{d\gamma} \left[ \frac{1}{2} (1,000 - (1,372 + \gamma))^2 + \frac{1}{2} (974 - (1,372 + \gamma))^2 \right] = 0$$

$$\gamma_{1,1} = \frac{(1,000 - 1,372) + (974 - 1,372)}{2} = -385$$

- We end up with the **average of the Residuals** in the leaf  $R_{1,1}$ , given our choice of Loss Function.

## Step 2.4) Update the model

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^J \gamma_{jm} I(x \in R_{jm}).$$

New prediction:

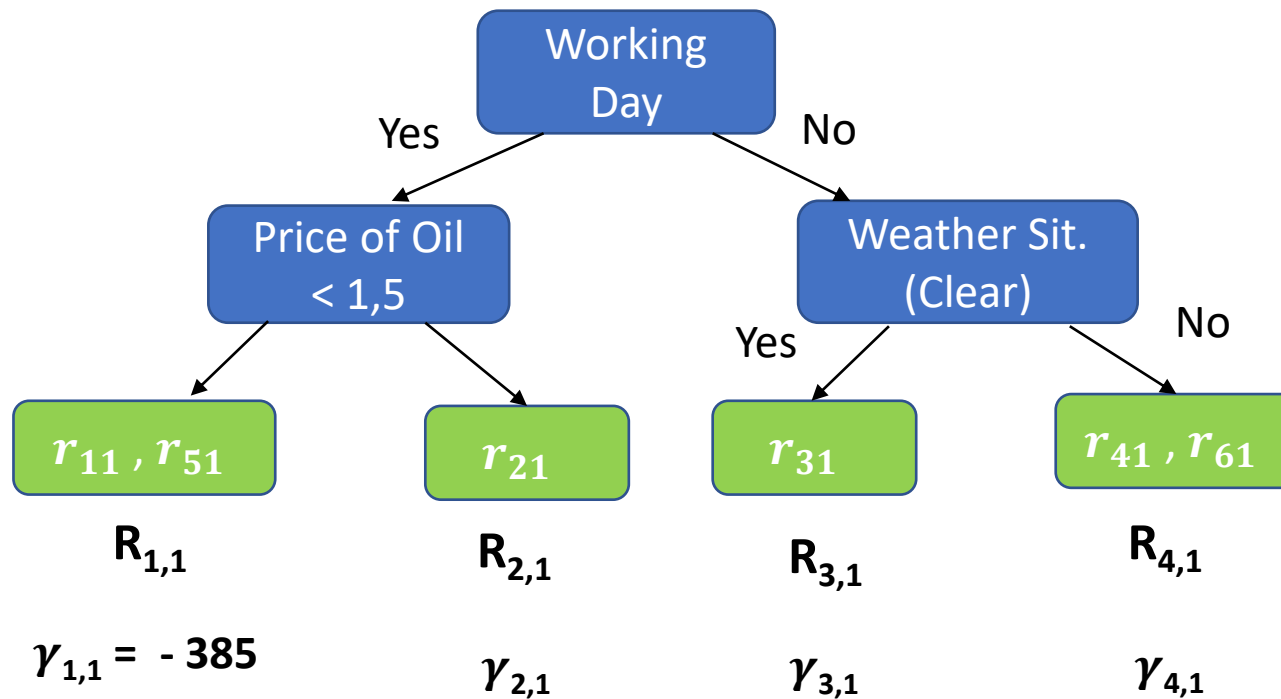
$$F_1(x) = F_0(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

- The summation means we add the **Output Values**,  $\gamma_{jm}$ , for all the leaves,  $R_{jm}$ , that a sample  $x$  can be found.
- $\nu$  is the **Learning Rate**. A small learning rate reduces the effect each tree has on the final prediction, and this improves accuracy of the final model, because it prevents overfitting. Range of parameter :  $0 < \nu < 1$



**New prediction for  $\mathbf{x}_1$ , with  $\nu = 0.1$ :**

$$F_1(\mathbf{x}_1) = F_0(\mathbf{x}_1) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(\mathbf{x} \in R_{jm}) = 1,372 + 0.1 \times (-385) = 1,333.5$$



We are closer to  
actual value (1,000)

We set  $m = 2$  and do everything over again:

2. For  $m = 1$  to  $M$ :

(a) For  $i = 1, 2, \dots, N$  compute

$$r_{im} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}.$$

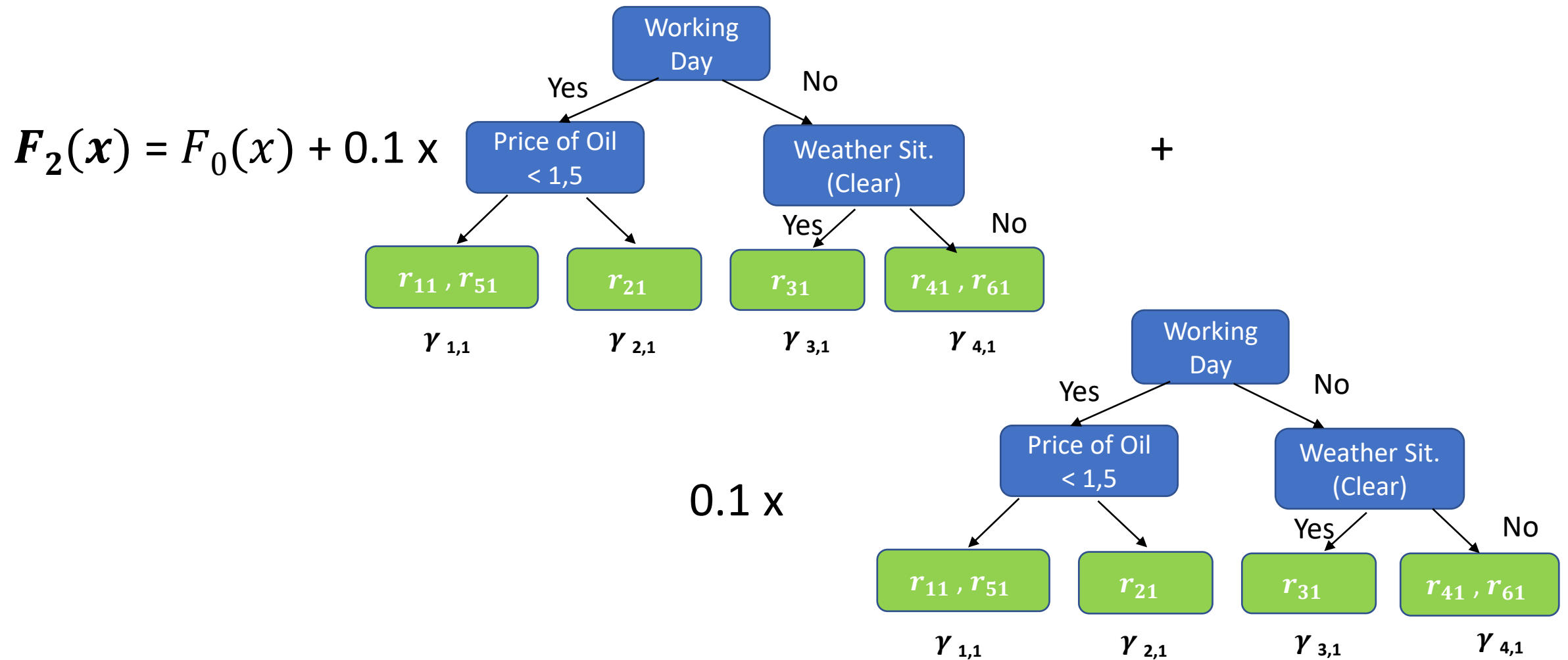
(b) Fit a regression tree to the targets  $r_{im}$  giving terminal regions  $R_{jm}$ ,  $j = 1, 2, \dots, J_m$ .

(c) For  $j = 1, 2, \dots, J_m$  compute

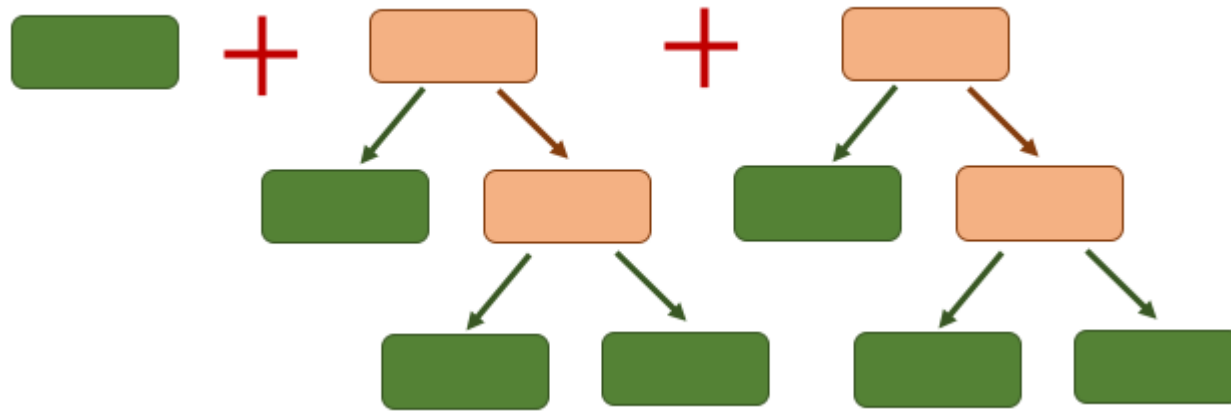
$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(d) Update  $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ .

In second loop,  $m = 2$ , the new predictions  $F_2(x)$  are:



Step 3) Output  $F_M(x)$  : our final predictive model



## Note:

- Number of trees,  $M$ , is setting by ourself. As we increase the number of componentes, the errors are reduced. However, fitting the training data too well can lead to overfitting.
- The number of trees must be balanced with the learning rate. A smaller learning rate may requires more trees, because small values of  $\nu$  result in larger training loss for the same number of iterations. In same sense a larger learning rate may requires fewer trees.
- Empirically, the model get better performance if the  $\nu$  is smaller, which requires large number of weak learners.
- Use a subsampling of training data at each iteration is another regularization strategy.

# Data Stream

- The algorithm was originally developed for static data, that is, the learning model is trained over a static, fixed size of data.
- The environment of data stream presents new challenges.
- One of the main challenges we can face is to deal with **non-stationary data**.
- In real world, typically, data stream evolves over time and it is affected by concept drift.
- To model and analyse dynamically arriving data, the original algorithm needs to suffer some adaptations.

# Concept Drift - Strategies

- Algorithms for handling concept drift can be divided into active and passive approaches
- **Active approaches** detect concept drift in every time step and react after confirming a drift
- For **passive methods**, the target is to learn data streams based on self-adjustment rather than rely on drift detection results.
- The ensemble models can be active or passive.
- Most of the current ensembles do not contain any drift detector and constantly update the model to adapt to change with new evolving data.

# Processing examples - Strategies

- In terms of processing examples that comes on the fly, there are two principal methods: Incremental learning and on-line learning.
- **Online learning:** learns a model when the training instances arrive sequentially one by one.
- **Incremental learning:** updates a model when a new batch of data instances arrive.
- The choice of method is independent of how the examples arrive (in chunks or separately).



# Typical incremental ensemble approach:

- A new base learner (component) is created from the newest chunk. It is added to ensemble model if the number of components is not exceeded; otherwise a member of ensemble is replaced by the newest component.
- The model is iteratively fitting from the most recent data and naturally adapting to drifts.
- The size of the chunk has important effects:
  - A small size could not be enough to learn well the entire data distribution, may result in poor performance, and would increase computational costs.
  - A large size may delay reaction to new concepts.
- The number of learners is also a crucial hyperparameter, because it influences the speed of the model to adapt to drifts.

# Active ensemble approach




**Advantage:** react quickly to the drift.

However, the model may be reacting to a false alarm. Therefore, the algorithm should be robust to false positive detections.

**Disadvantage:** increase computational resources and time.

**Trade-off** between computational costs and gains in performance.

# Implementations of Gradient Boosting for Data Streams

- Online Gradient Boosting (2015) by A. Beygelzimer et al.
- Gradient Boosting on Stochastic Data Streams (2017) by H. Hu et al.
- An Elastic Gradient Boosting Decision Tree for Concept Drift Learning" (2020) by K. Wang 
- On Incremental Learning for Gradient Boosting Decision Trees" (2019) by C. Zhang et al. 
- Adaptive XGBoost for Evolving Data Streams" (2020) by J. Montiel et al. 
- Federated Soft Gradient Boosting Machine for Streaming Data" (2020) by J. Feng et al.

# Elastic Gradient Boosting Decision Tree (eGBDT)

- Construct M tree members in a iterative way.
  - Assign  $f_0(x)$  based on initial chunk
  - Compute the pseudo-residuals
  - Fit a tree to the negative gradient of the loss.
  - Update the model
  - Calculate the pseudo-residuals and add a fitted tree consecutively until have M components.
- When a new chunk of data arrives, the algorithm add L number of trees in the same way described before.

# Elastic Gradient Boosting Decision Tree (eGBDT)

- Compute the residuals for each length of the model, i.e., for each  $m$ th tree.

$$R_m = y - F_m(x) = y - \bar{y} - \sum_{i=1}^m h_i(x),$$

- Find the sub tree with a best performance, selecting the sub tree with lowest mean absolute (MA) error.

$$I_{\text{elastic}} = \underset{m \in \mathbb{Z}_{m \leq M}^+}{\operatorname{argmin}} \operatorname{MA}(R_m)$$

- eGBDT will maintain the trees until  $I_{\text{elastic}}$  tree index  $\{h_0(x), \dots, h_{I_{\text{elastic}}}(x)\}$ , and discard the remaining trees,  $\{h_{I_{\text{elastic}}+1}(x), \dots, h_M(x), \}$ .

# Elastic Gradient Boosting Decision Tree (eGBDT)

- If  $l_{elastic}$  is smaller than a predefined threshold, e.g.,  $M$  tree components, we consider that there is a concept drift, then GBDT will be retrained on the new data.
- If not, the algorithm continues fit the GBDT with  $L$  new trees on the new data chunk.

# Elastic Gradient Boosting Decision Tree (eGBDT)

## Case 1

- If the best prediction occurs at the 80th tree, when the initial training has 100 trees only, then the GBDT model can not adapt to the data. This is a significant drift and the model is retrained.

## Case 2

- Incrementally fine tune learning 50 more trees. If the best prediction occurs at the 150th tree, this is probable underfitting. If the best prediction occurs at the 110th tree, the model is probably overfitting and the trees after the 110th trees are deleted.

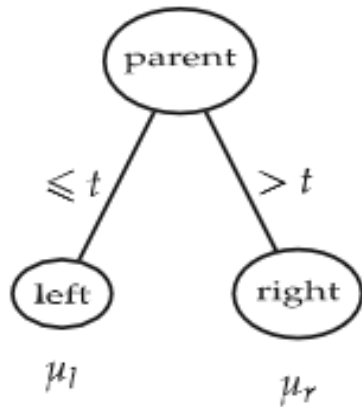
# Incremental Gradient Boosting Decision Tree (iGBDT)

- iGBDT focuses on reducing computational time.
- It incrementally learns a new model when a new batch of data instances arrive without running GBDT from scratch. iGBDT aims at “lazily” updating the old GBDT ensemble model.
- Gives significantly better performance in terms of model building/updating time than GBDT of the same accuracy.
- Important for timely analysis of continuously arriving or real-time user generated data e.g. behaviour targeting, internet advertising, recommender systems, etc.



# iGBDT Algorithm

1. Builds GBDT for the first chunk of data by finding the best split attribute and value



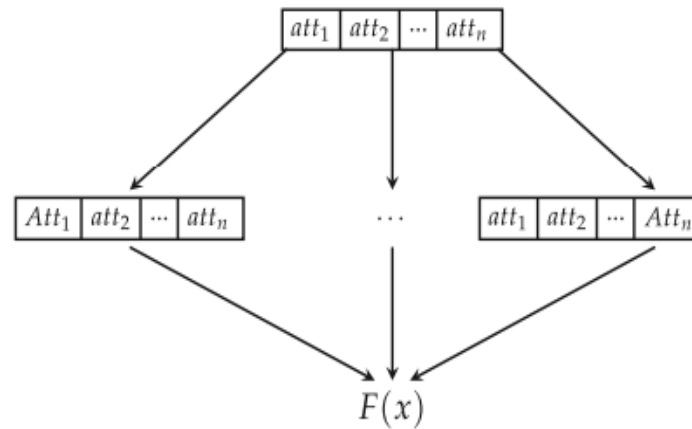
$$\mu_l = \frac{1}{l} \sum_{i=1}^l y_i \quad \mu_r = \frac{1}{r} \sum_{j=1}^r y_j$$

$$\operatorname{argmin} \left[ \sum_{i=1}^l (y_i - \mu_l)^2 + \sum_{i=1}^r (y_i - \mu_r)^2 \right]$$

- Instead of use information Gain, for each attribute, it pick the split value that achieves the minimum sum square loss error.
- The attribute with lowest sum square loss error is chosen as the best split attribute.

# iGBDT Algorithm

- Each attribute of the train data set is sorted in ascending order

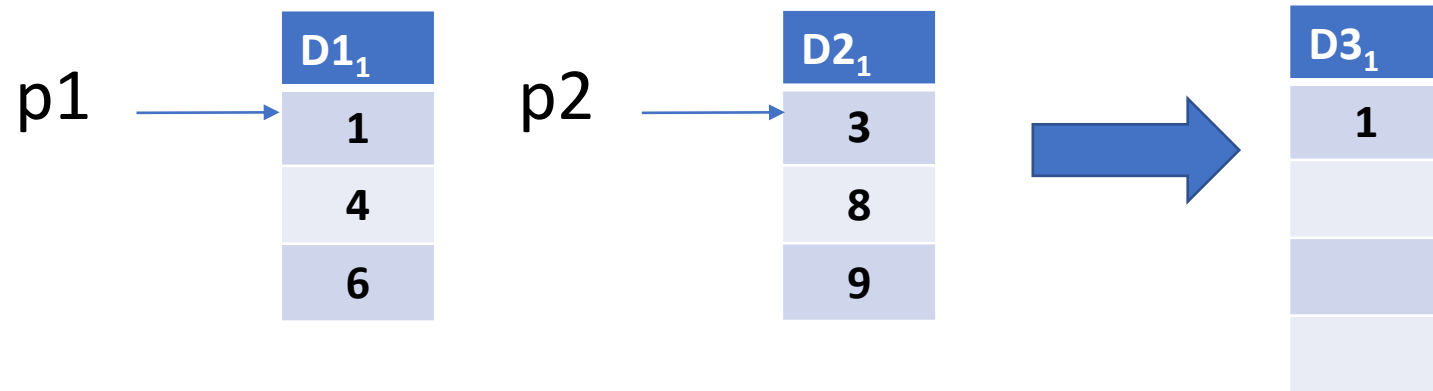


- In iGBDT, the previously sorted data set for each attribute will be reused by all the decision trees, thus this sorting is an once-for-all computation process in iGBDT.

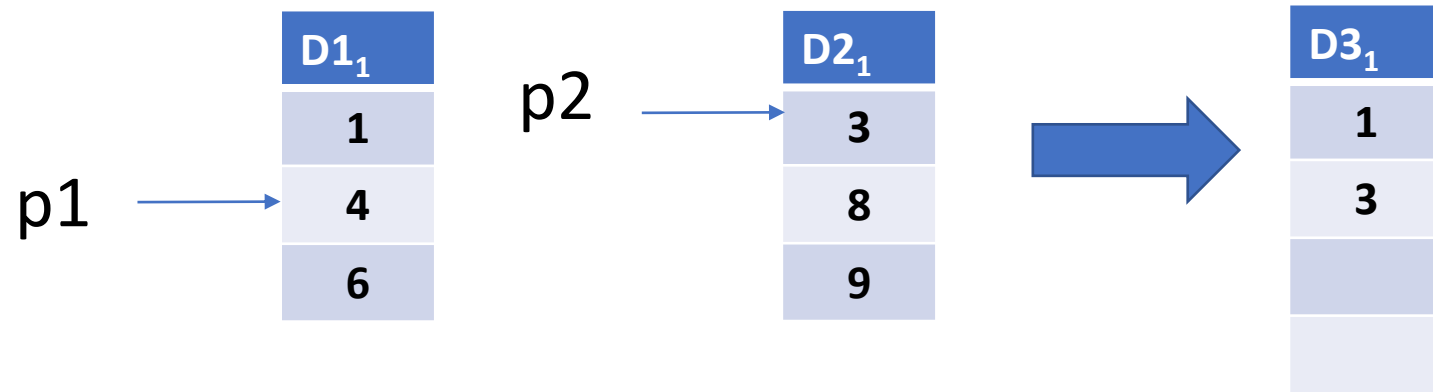
# iGBDT Algorithm

2. Apply the “sort-merge” method to speed up the sorting process
  - Merges the original data D1 and the new batch data D2 (both ordered according to attribute k) to obtain an ordered data set D3. So the number of samples in  $D3 = D2 + D1$
  - Sets two pointers, p1 which initially points to the first element of the originally sorted set, and p2 points to the sorted list of the new batch for the same attribute and compares p1 and p2 in the following manner

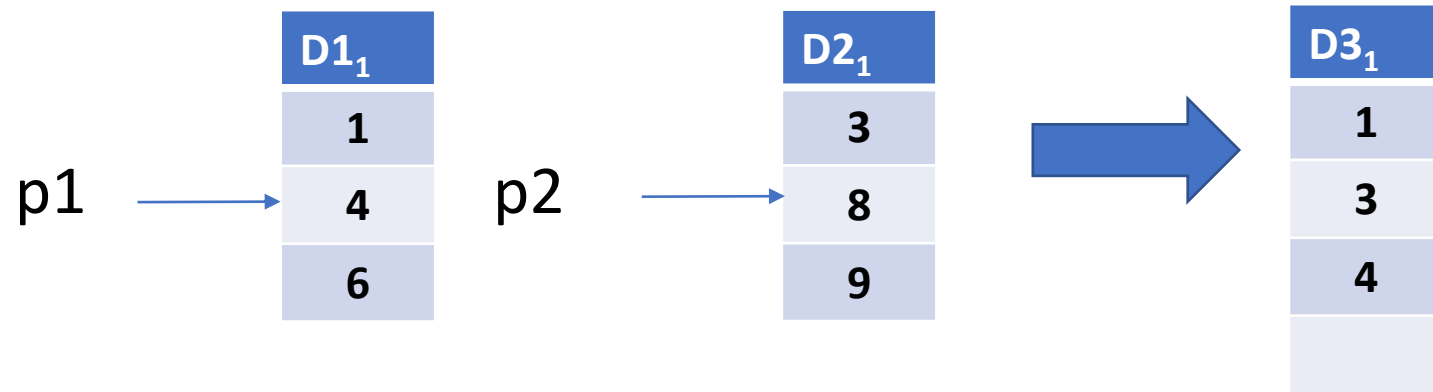
# iGBDT Algorithm



# iGBDT Algorithm



# iGBDT Algorithm



With the “sort-merge” method, iGBDT only needs to scan two ordered sets once, thus it greatly reduces the amount of time needed in re-sorting the whole data, especially when the original ordered set is much larger than the new batch of data.

# iGBDT Conclusion

**3.** Timely checks and updates previously built GBDT using the newly re-ordered data

- Sequentially checks each decision tree model.
- Starting with the first tree, finds best split attribute (and the corresponding best split point/value) for the newly ordered data, then checks if this attribute is the same as the one on the root node of the first decision tree.
- If yes, then iGBDT only updates the auxiliary information of the node, mainly the mean class value and best split point. Then, it horizontally splits the newly ordered data into left and right parts.

# iGBDT Conclusion

- iGBDT checks each decision tree and its sub-trees to figure out whether the corresponding best split attribute has changed, but it saves the huge amount of time spent on rebuilding the thousands of decision trees
- Online BBM outperforms iGBDT in training time, while iGBDT beats Online BBM in both accuracy results of prediction efficiency
- VFDT is faster than iGBDT in training time, in terms of prediction efficiency, iGBDT outperforms VFDT
- It must be emphasized that, these three algorithms fit to different application scenarios



# iGBDT Conclusion

- The total running time of iGBDT on  $|N_b|$  batches would be significantly larger than when directly run on Data set D.
- Incremental learning fits to application scenarios where the batch of new data arrives on the fly, i.e. the subsequent batches of data are unavailable when processing the current batch of new data, and also there are constraints for in-memory processing, i.e. total RAM is inadequate to hold all batches of data
- The performance of iGBDT also depends on the data characteristics; if the concept evolves rapidly, iGBDT's efficiency will drop consequently.

# XGBoost

- XGBoost stands for **Extreme Gradient Boosting**.
- XGBoost is a specific **implementation of the Gradient Boosting** method which delivers more accurate approximations by using the strengths of **second order derivative of the loss function, L1 and L2 regularization and parallel computing**.

# XGBoost vs. Standard Gradient Boosting

- XGBoost is more **regularized form of Gradient Boosting**. It uses advanced regularization (L1 & L2), which improves model generalization capabilities.
- XGBoost delivers high performance as compared to Gradient Boosting. Its training is very fast and can be **parallelized / distributed across clusters**.

# Adaptive XGBoost (AXGB) for Data Streams

- Method proposed in 2020 by **Jacob Montiel**, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem and Albert Bifet to perform **classification** on evolving data streams;
- To adapt the XGBoost to the streaming world it uses a **buffer**, 2 possible **ensemble update strategies** and **drift detection**.

# AXGB - Ensemble

- The ensemble is created using **forward additive modeling**. The training data is evaluated on existing members of the ensemble and the corresponding prediction scores  $\hat{Y}^{(k)}$  are used to train new members of the ensemble. The base functions predictions are combined additively:

$$\hat{Y}^{(k)} = \sum_{k=1}^K f_k(X) = \hat{Y}^{(k-1)} + f_k(X)$$

# AXGB - Predictions

- The final prediction for a sample is the sum of the predictions for each tree  $f_k$  in the ensemble.

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i)$$

# AXGB – Mini Batches

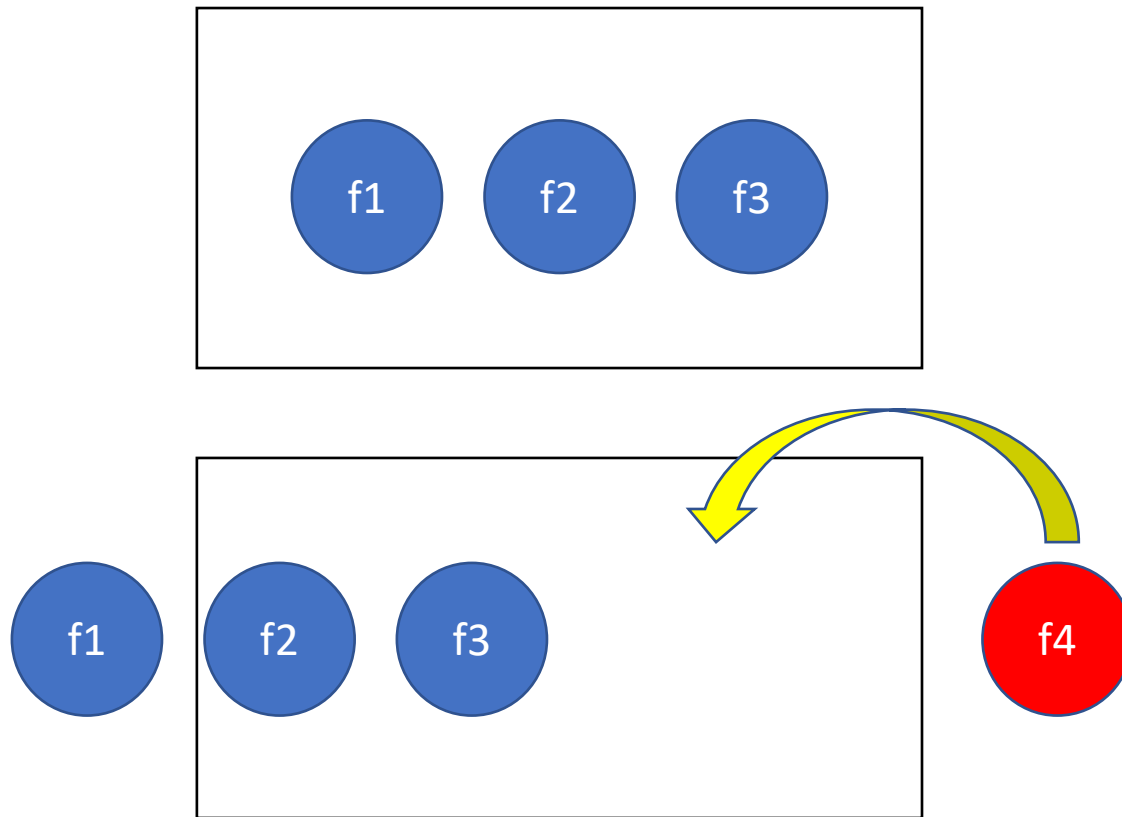
- The proposed method creates new members of the **ensemble** from "**mini-batches**" of data as new data becomes available;
- The mini-batches are obtained using non overlapping windows;
- The size of the windows (buffer) increases exponentially until it reaches a pre-defined maximum.

# AXGB – Ensemble Update

- When the ensemble reaches the pre-defined maximum (K) size there are two possible strategies: **Push** or **Replace** to continue training and evolving the model;
- Note that in both strategies we have to wait K iterations to get a completely new ensemble;



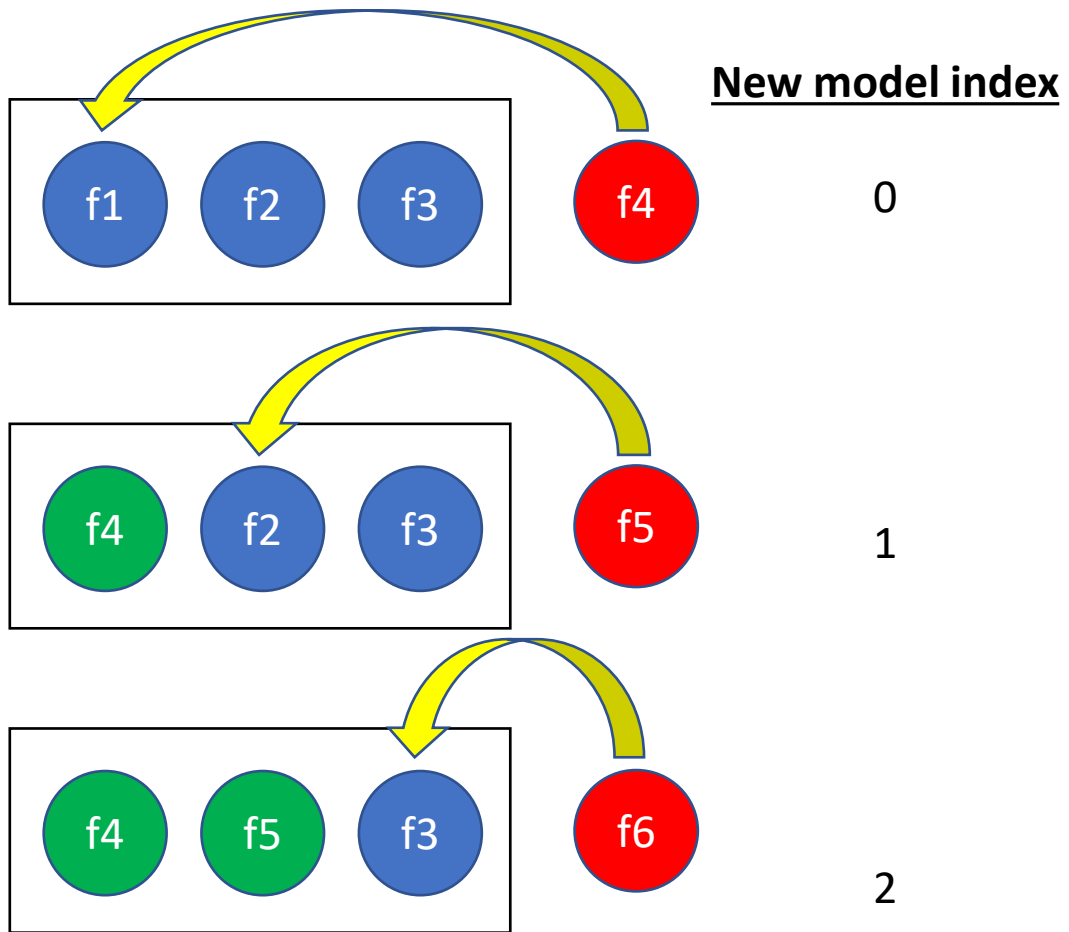
# AXGB - Update: Push



The ensemble is full. To keep it updated **Push** can be applied.

When the buffer is full a new booster (f4) is trained using the residuals from the models in the ensemble (f1, f2, f3). The oldest model f1 is "pushed" out.

# AXGB - Update: Replace



The ensemble is full. When the buffer reaches its capacity an index is initialized at 0. However, f4 is not trained using the residuals from the previous models.

In the next update, the new booster is trained using the residuals from the models that are on the left from the current index. f5 is trained with f4 residuals.

Finally, f6 is trained using residuals from f4 and f5.

# AXGB – Drift Detection

ADWIN is used to perform drift detection as new data comes;

When ADWIN detects change, window size is reset and in  
Replace the new model index is also resetted;

# AXGB – Conclusions

- Authors concluded that AXGB[r] (the version that performs model replacement in the ensemble but does not include explicit concept drift awareness) represents the best compromise in terms of performance, training time and memory usage;
- AXGB[p] is slightly slower in training because it uses more ensemble members to train new learners;
- Using drift detection also has a negative impact on the training times;

# AXGB – Conclusions

- Regarding AXGB[p] enabling drift detection has a positive impact in predictive performance;
- Meanwhile AXGB[r] shows little to no benefit in using ADWIN;
- The learning rate has a consistent impact on performance (lower is better), followed by max window size and max depth.

# Conclusion

- Gradient boosting is a very powerful technique, expanding it to the data streaming world further amplifies its strength;
- In this presentation we shown theoretical aspects related to it and also some implementations;

# Bibliography

- [1]J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, ‘Adaptive XGBoost for Evolving Data Streams’, arXiv:2005.07353 [cs, stat], May 2020, Accessed: Dec. 05, 2021. [Online]. Available: <http://arxiv.org/abs/2005.07353>
- [2]K. Wang, A. Liu, J. Lu, G. Zhang, and L. Xiong, ‘An Elastic Gradient Boosting Decision Tree for Concept Drift Learning’, in AI 2020: Advances in Artificial Intelligence, Cham, 2020, pp. 420–432.
- [3]B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, ‘Ensemble learning for data stream analysis: A survey’, Information Fusion, vol. 37, pp. 132–156, Sep. 2017, doi: 10.1016/j.inffus.2017.02.004.
- [4]J. Feng, Y.-X. Xu, Y.-G. Wang, and Y. Jiang, ‘Federated Soft Gradient Boosting Machine for Streaming Data’, in Federated Learning: Privacy and Incentive, Q. Yang, L. Fan, and H. Yu, Eds. Cham: Springer International Publishing, 2020, pp. 93–107. doi: 10.1007/978-3-030-63076-8\_7.
- [5]H. Hu, W. Sun, A. Venkatraman, M. Hebert, and J. A. Bagnell, ‘Gradient Boosting on Stochastic Data Streams’, p. 9.
- [6]C. Zhang, Y. Zhang, X. Shi, G. Almpanidis, G. Fan, and X. Shen, ‘On Incremental Learning for Gradient Boosting Decision Trees’, Neural Process Lett, vol. 50, no. 1, pp. 957–987, Aug. 2019, doi: 10.1007/s11063-019-09999-3.
- [7]A. Beygelzimer, E. Hazan, S. Kale, and H. Luo, ‘Online Gradient Boosting’, p. 9. [8]S. Tibshirani, H. Friedman, and T. Hastie, ‘The Elements of Statistical Learning’, p. 764, 2017.