Imersão em Ciência de Dados 2019

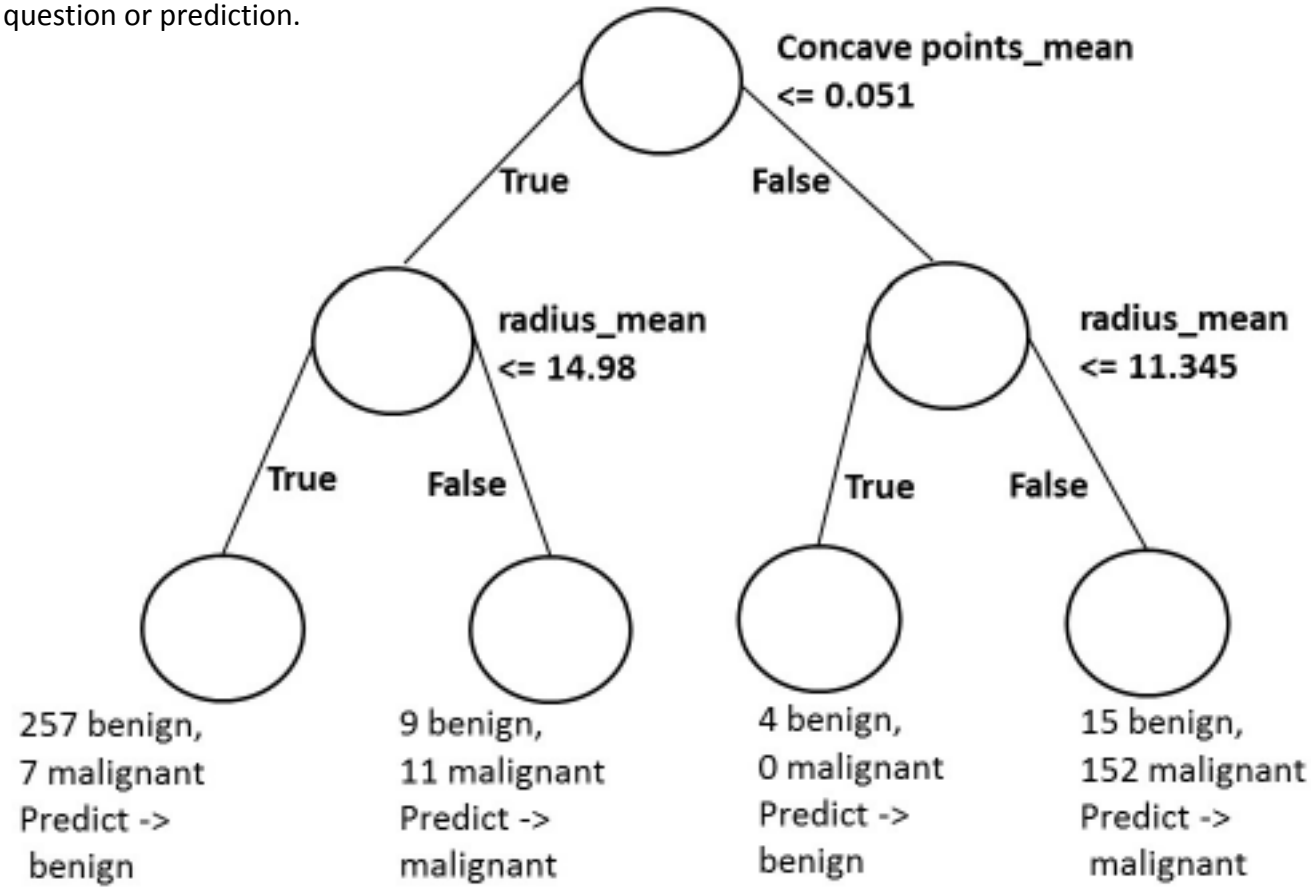
Fortaleza (CE), 15 a 20 de julho de 2019.

Decision Trees

- Sequence of if-else questions about individual features.
- Data structure consisting of a hierarchy of nodes.
- **Goal**: infer class labels.
- Capture non-linear relationships between features and labels.
- Don't require feature scaling (ex: Standardization, ..)

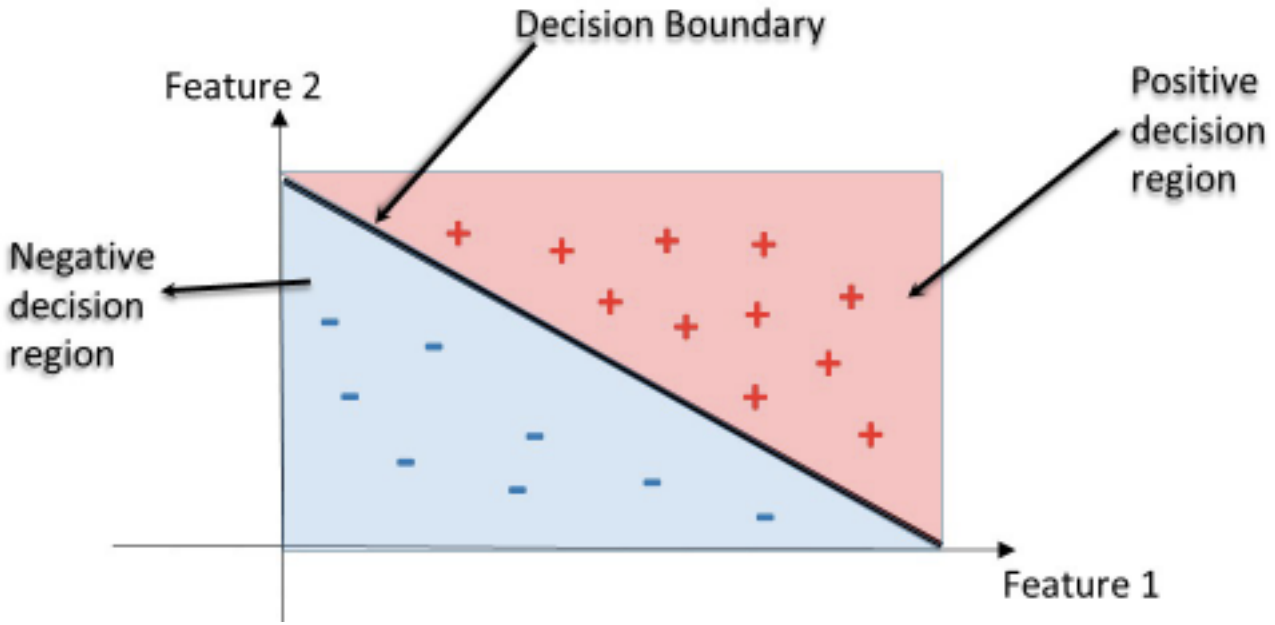
Diagram

Node: question or prediction.

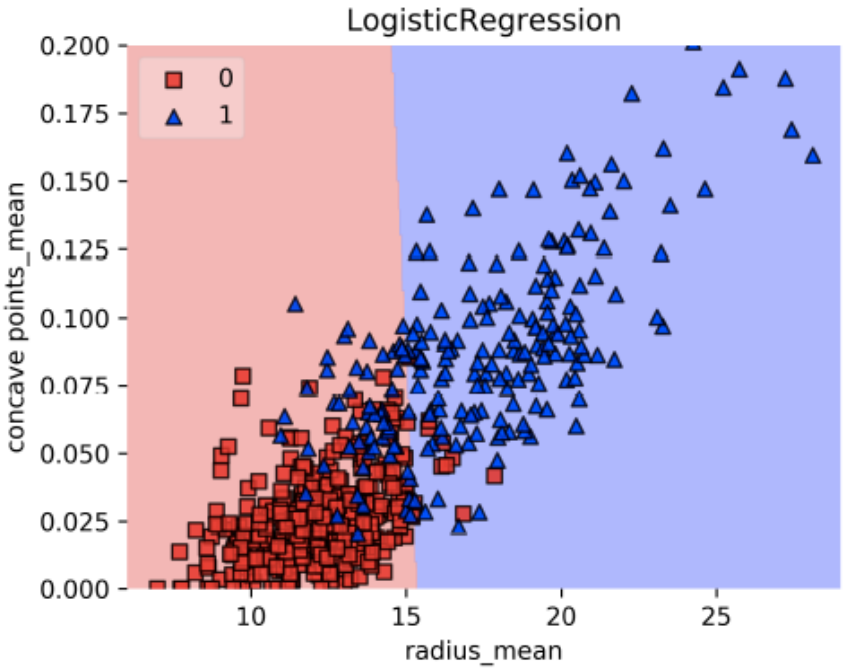


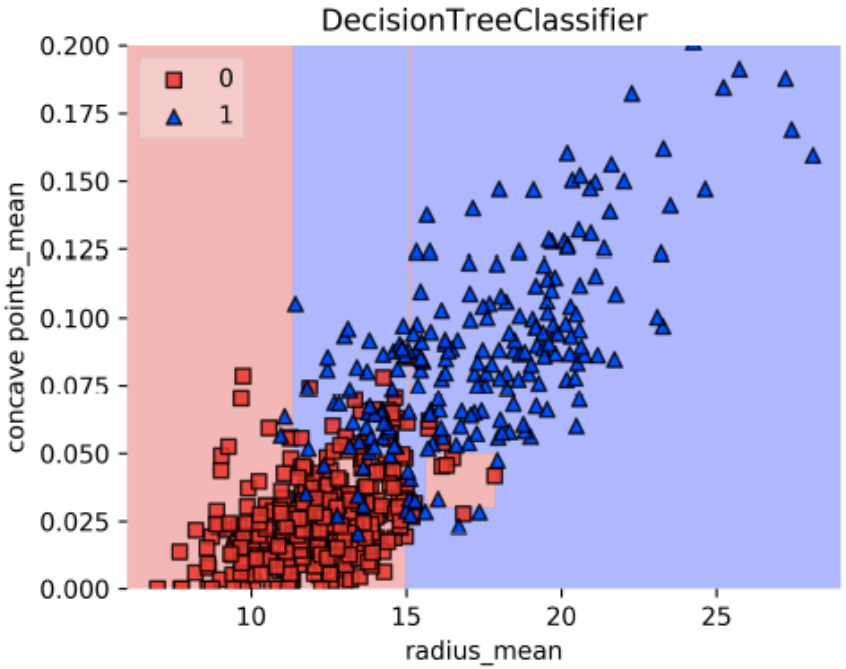
Decision Regions

- Region in the feature space where all instances are assigned to one class label.
- **Decision Boundary**: surface separating different decision regions

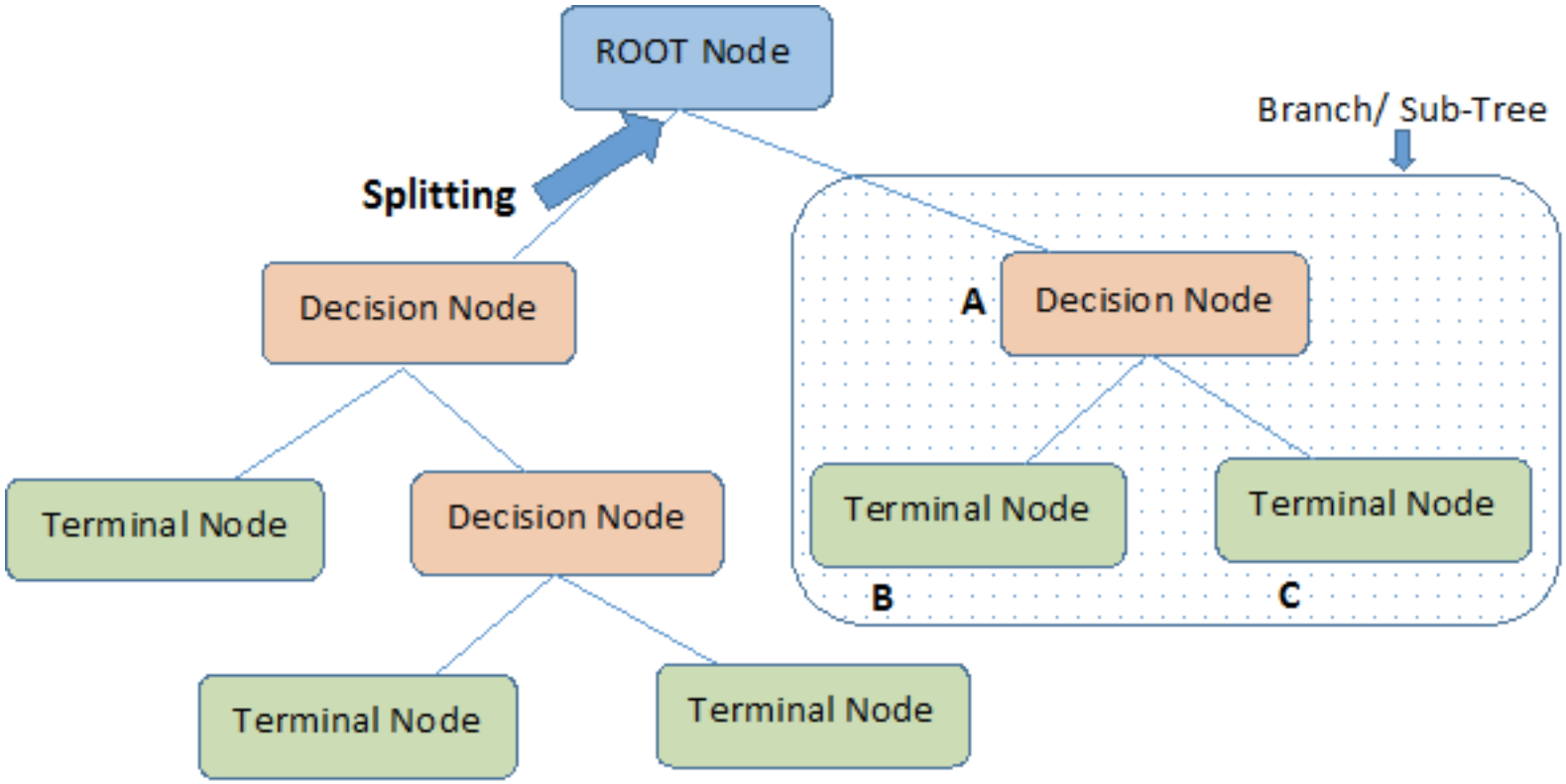


Decision Regions





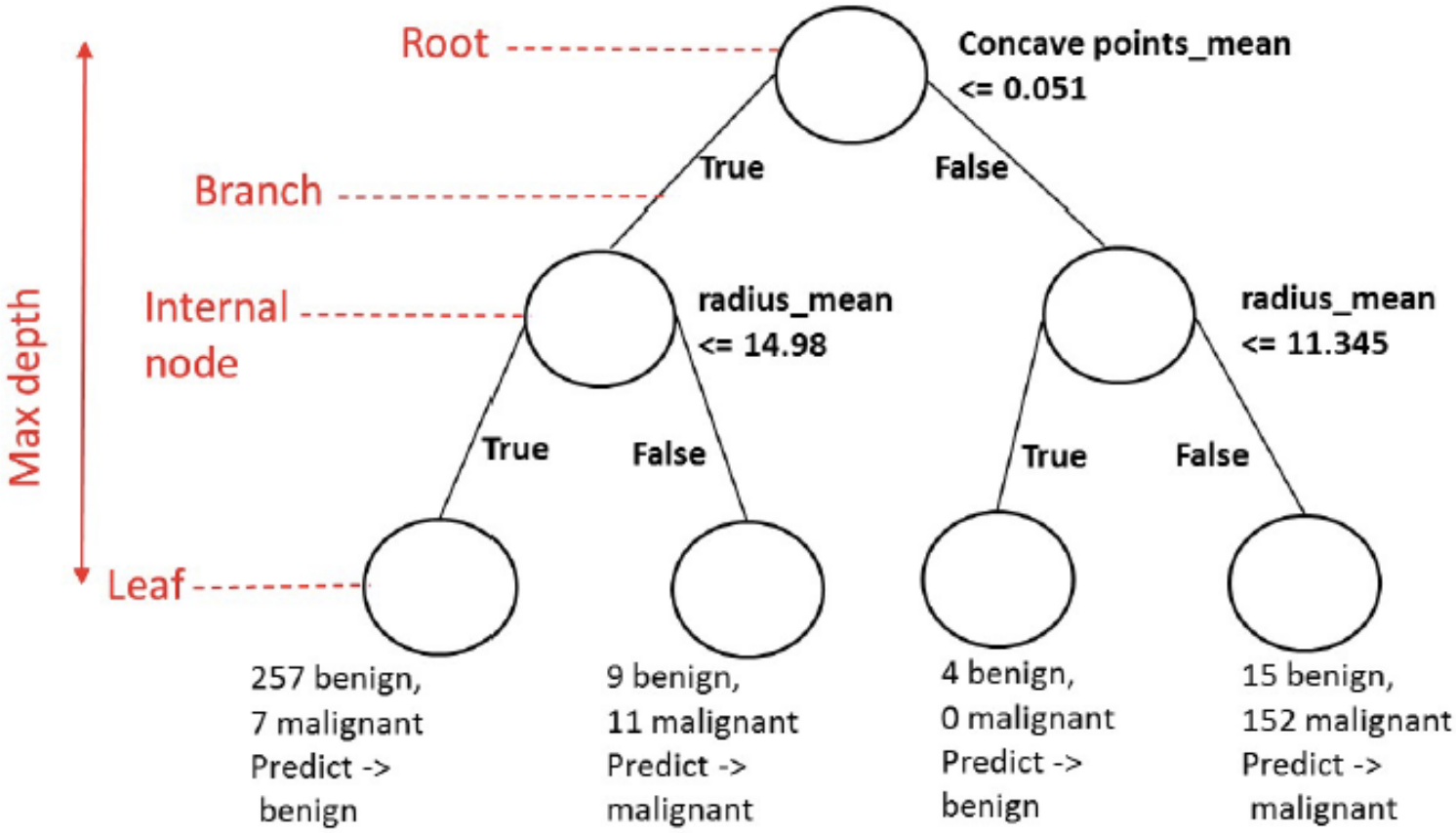
Terminology



Note:- A is parent node of B and C.

Tree building starts by finding the variable/feature for best split.

Decision Tree Prediction



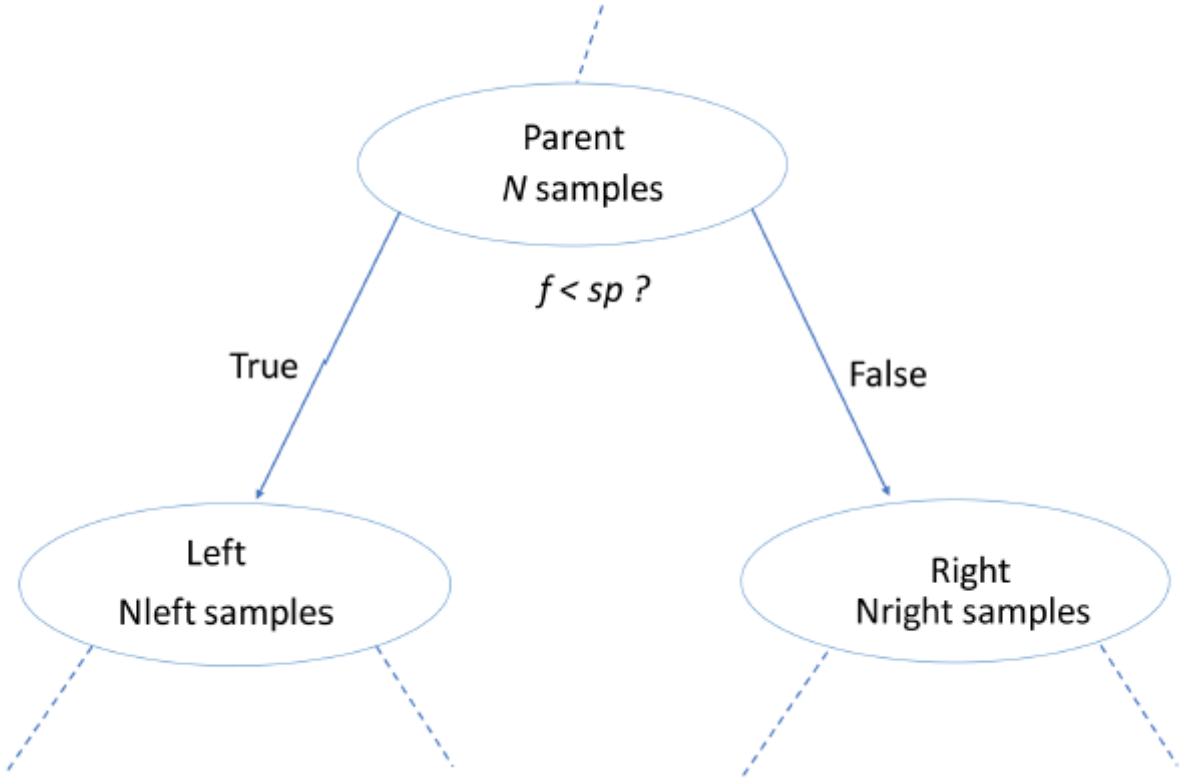
Terminology

- **Root Node:** Entire population or sample, further gets divided into two or more homogeneous sets.
- **Parent and Child Node:** Node which is divided into sub-nodes is called parent node, whereas sub-nodes are the child of parent node.
- **Splitting:** Process of dividing a node into two or more sub-nodes.
- **Decision Node:** A sub-node that splits into further sub-nodes.
- **Leaf/ Terminal Node:** Nodes that do not split.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. (Opposite of Splitting)
- **Branch/Sub-Tree:** Sub-section of entire tree.

Decision tree learning

- We start at the tree root and split the data on the feature that results in the largest Information Gain (IG).
- We can then repeat this splitting procedure at each child node until the leaves are pure.
- The samples at each node all belong to the same class.
- This can result in a very deep tree with many nodes, which can easily lead to overfitting.
  - Thus, we typically want to prune the tree by setting a limit for the maximal depth of the tree.
    - Hyper-parameter: max_depth.

Information Gain (IG)



f	feature
sp	split-point

Information Gain (IG)

$$IG(\underbrace{f}_{feature}, \underbrace{sp}_{split-point}) = I(parent) - \left(\frac{N_{left}}{N} I(left) + \frac{N_{right}}{N} I(right)\right)$$

I(node)	Impurity measure of a node
N(node)	Total number of samples of a node

Funcionamento

- Nodes are grown recursively.

- At each node, split the data based on:
  - feature *f* and split-point *sp* to maximize *IG*(node).

- If *IG*(node)= 0, declare the node a leaf.

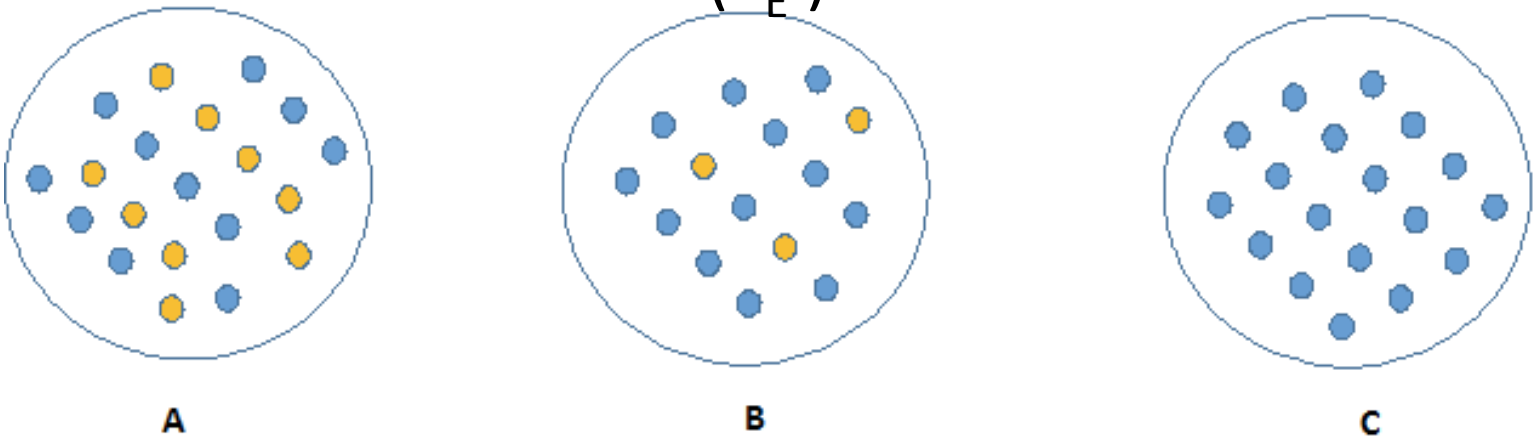
Maximizing information gain

- Information Gain (IG) is:
  - the difference between the impurity of the parent node and the sum of the child node impurities.
  - The lower the impurity of the child nodes, the larger the information gain.
- For simplicity and to reduce the combinatorial search space, most libraries implement binary decision trees.
  - Each parent node is split into two child nodes, D_{left} and D_{right} :

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

Impurity Measures

- 3 Impurity measures or splitting criteria commonly used in binary decision trees:
  - Gini Impurity (I_G)
  - Entropy (I_H)
  - Classification Error (I_E)



Entropia

- Medida da quantidade de desordem de um sistema.
- Entropy is also a measure of the number of possible arrangements the atoms in a system can have.
  - In this sense, entropy is a measure of uncertainty or randomness.
  - The higher the entropy of an object, the more uncertain we are about the states of the atoms making up that object because there are more states to decide from.

Entropy

$$I_H(t) = -\sum_{i=1}^{c} p(i \mid t) \log_2 p(i \mid t)$$

- Entropy attempts to maximize the mutual information in the tree.
- **Measure of randomness.** More the random data, higher the entropy.
- p (i | t) is the proportion of the samples that belong to class c for a particular node t.
  - Zero - if all samples at a node belong to the same class.
    - Binary class example:
      - p (i = 1 | t) = 1 or p (i = 0 | t) = 0
  - Maximal (1) - if we have a uniform class distribution.
    - Binary class example:
      - p (i = 1 | t) = 0.5 and p (i = 0 | t) = 0.5

Gini

- Criterion to minimize the probability of misclassification:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)\left(1 - p(i|t)\right) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

- **Measure of variance** across all classes of the data.
- In practice both Gini impurity and entropy typically yield very similar results.
  - So it is often not worth spending much time on evaluating trees using different impurity criteria

Classification Error

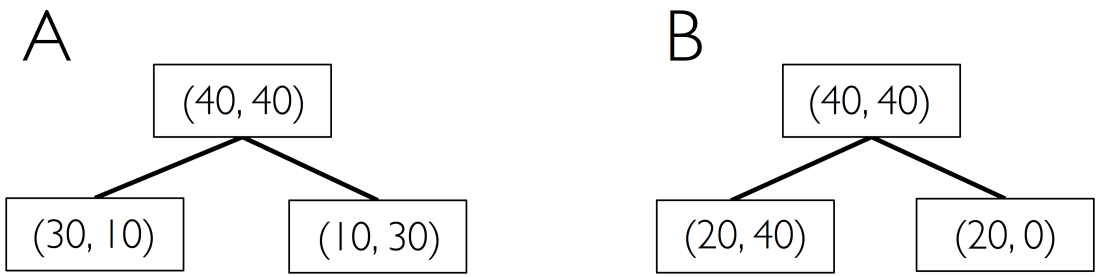
- Useful criterion for pruning, but not recommended for growing a decision tree, since it is less sensitive to changes in the class

$$I_E = 1 - \max\{p(i \mid t)\}$$

es.

Example - IG_E

$$I_E = 1 - \max\{p(i|t)\}$$



- D_p consists in 40 samples from class 1 and 40 samples from class 2 that we split into two datasets, D_{left} and D_{right}.

$$I_E(D_p) = 1 - 0.5 = 0.5$$

$$A: I_E(D_{left}) = 1 - \frac{3}{4} = 0.25$$

$$A: I_E(D_{right}) = 1 - \frac{3}{4} = 0.25$$

$$A: IG_E = 0.5 - \frac{4}{8}0.25 - \frac{4}{8}0.25 = 0.25$$

$$B: I_E(D_{left}) = 1 - \frac{4}{6} = \frac{1}{3}$$

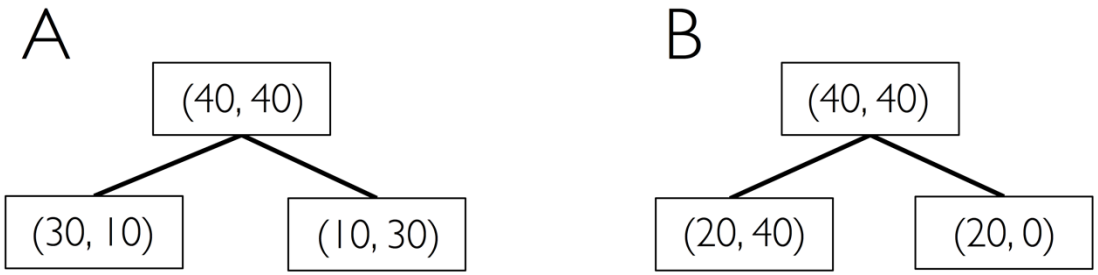
$$B: I_E(D_{right}) = 1 - 1 = 0$$

$$B: IG_E = 0.5 - \frac{6}{8} \times \frac{1}{3} - 0 = 0.25$$

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

Example - IG_G

$$I_G(t) = 1 - \sum_{i=1}^{c} p(i|t)^2$$



- D_p consists in 40 samples from class 1 and 40 samples from class 2 that we split into two datasets, D_{left} and D_{right}.

$$I_G(D_p) = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$A: I_G(D_{left}) = 1 - \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right) = \frac{3}{8} = 0.375$$

$$A: I_G(D_{right}) = 1 - \left(\left(\frac{1}{4}\right)^2 + \left(\frac{3}{4}\right)^2\right) = \frac{3}{8} = 0.375$$

$$A: IG_G = 0.5 - \frac{4}{8}0.375 - \frac{4}{8}0.375 = 0.125$$

$$B: I_G(D_{left}) = 1 - \left(\left(\frac{2}{6}\right)^2 + \left(\frac{4}{6}\right)^2\right) = \frac{4}{9} = 0.\overline{4}$$

$$B: I_G(D_{right}) = 1 - (1^2 + 0^2) = 0$$

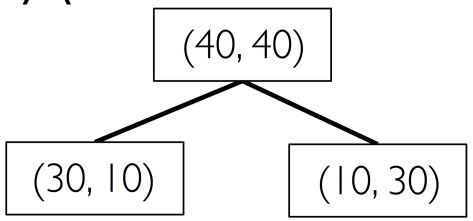
$$B: IG_G = 0.5 - \frac{6}{8}0.\overline{4} - 0 = 0.1\overline{6}$$

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

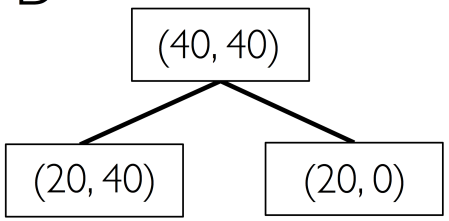
Example - IG_H

$$I_H(t) = -\sum_{i=1}^{c} p(i|t)\log_2 p(i|t)$$

A



B



- D_p consists in 40 samples from class 1 and 40 samples from class 2 that we split into two datasets, D_{left} and D_{right}.

$$I_H(D_p) = -(0.5\log_2(0.5) + 0.5\log_2(0.5)) = 1$$

$$A: I_H(D_{left}) = -\left(\frac{3}{4}\log_2\left(\frac{3}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right) = 0.81$$

$$A: I_H(D_{right}) = -\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right) = 0.81$$

$$A: IG_H = 1 - \frac{4}{8}0.81 - \frac{4}{8}0.81 = 0.19$$

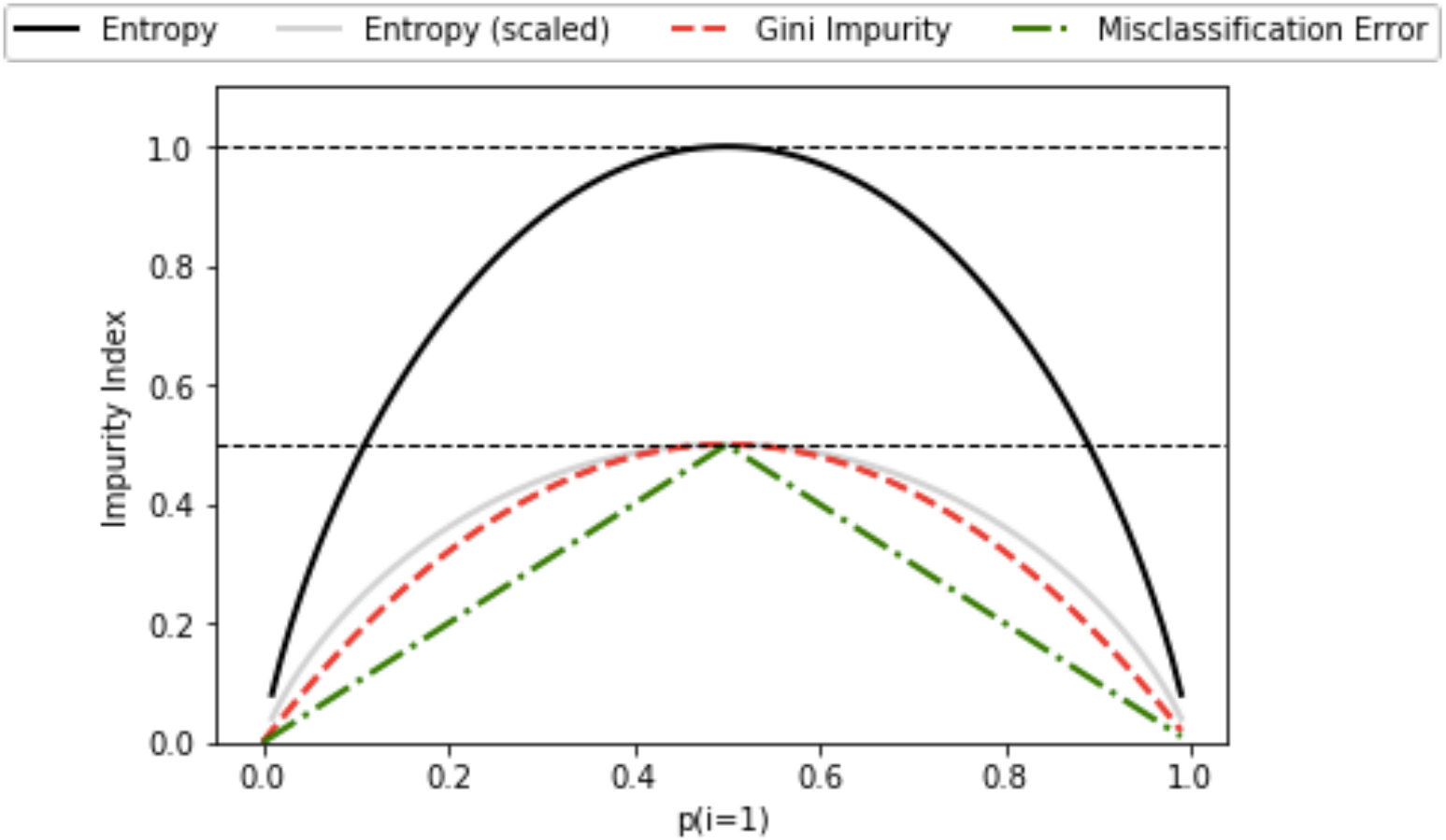
$$B: I_H(D_{left}) = -\left(\frac{2}{6}\log_2\left(\frac{2}{6}\right) + \frac{4}{6}\log_2\left(\frac{4}{6}\right)\right) = 0.92$$

$$B: I_H(D_{right}) = 0$$

$$B: IG_H = 1 - \frac{6}{8}0.92 - 0 = 0.31$$

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

Comparison of Impurity Measures



Comparison of Impurity Measures

- Most of the time, the gini index and entropy lead to the same results.

- The gini index is slightly faster to compute and is the default criterion used in the DecisionTreeClassifier model of scikit-learn.

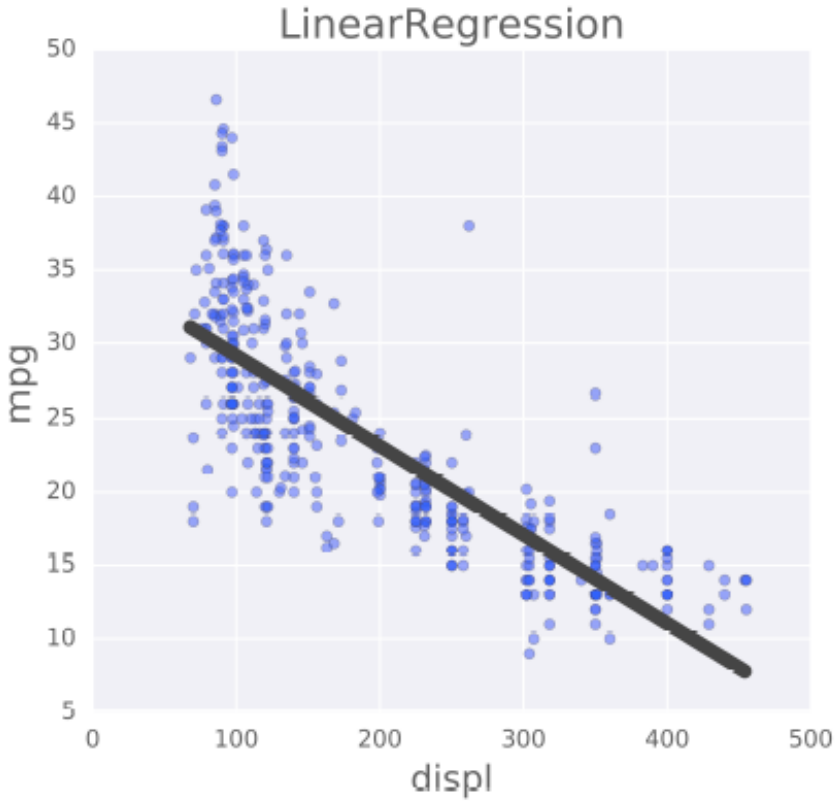
Regression-Tree

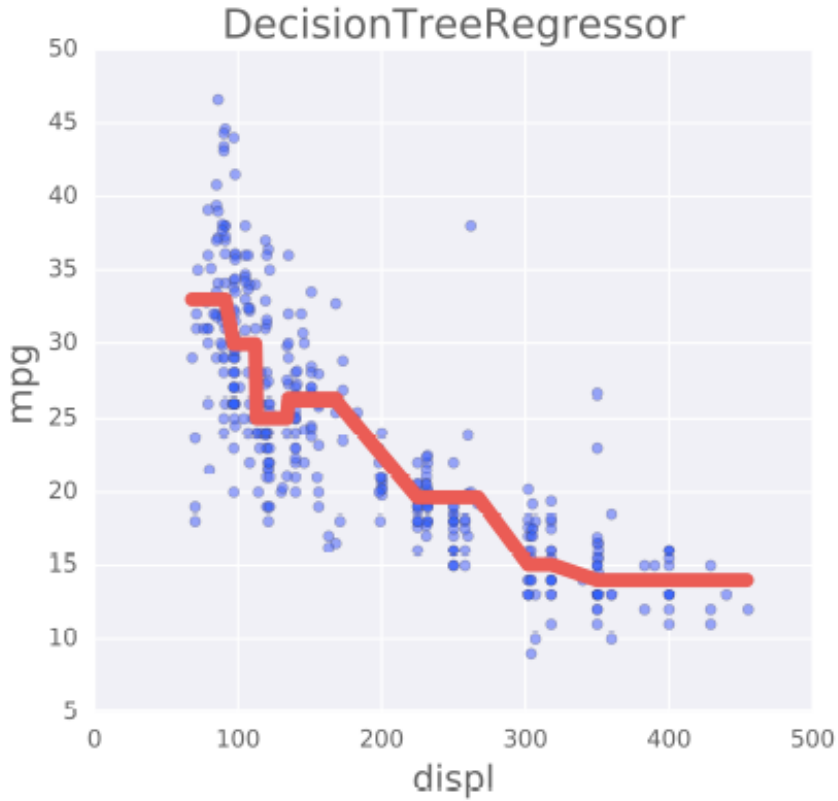
$$I(\text{node}) = \underbrace{\text{MSE(node)}}_{mean-squared-error} = \frac{1}{N_{node}} \sum_{i \in node} \left(y^{(i)} - \hat{y}_{node}\right)^2$$

$$\underbrace{\hat{y}_{node}}_{mean-target-value} = \frac{1}{N_{node}} \sum_{i \in node} y^{(i)}$$

$$\hat{y}_{pred}(\text{leaf}) = \frac{1}{N_{leaf}} \sum_{i \in leaf} y^{(i)}$$

Linear Regression vs Regression-Tree

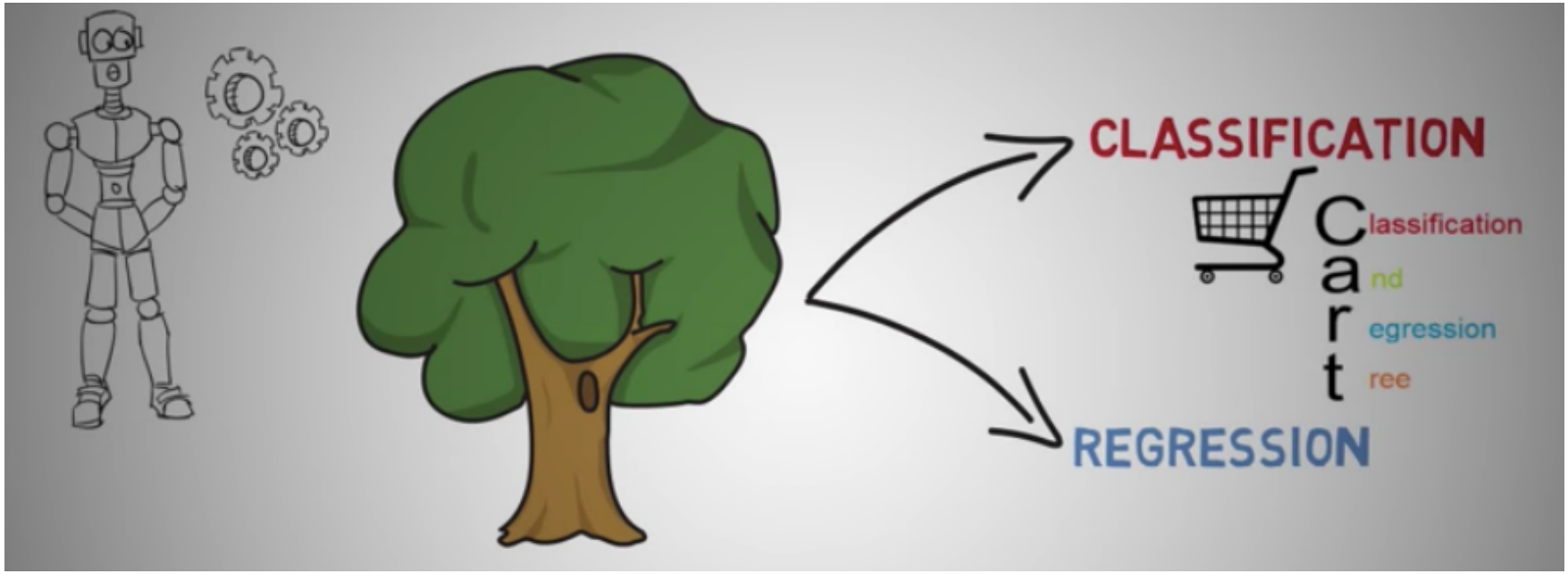




Decision-tree algorithms

- ID3 - Iterative Dichotomiser 3
- C4.5 - successor of ID3
- CART - Classification And Regression Tree
- CHAID - CHi-squared Automatic Interaction Detector
  - Performs multi-level splits when computing classification trees.
- MARS - multivariate adaptive regression splines
  - extends decision trees to handle numerical data better.
- Conditional Inference Trees.
  - Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting.
  - This approach results in unbiased predictor selection and does not require pruning.
- ID3 and CART were invented independently at around the same time (between 1970 and 1980) and follow a similar approach for training.

CART – Classification and Regression Tree



https://www.youtube.com/watch?v=DCZ3tsQIoGU

CART – Classification and Regression Tree

https://www.youtube.com/watch?v=DCZ3tsQIoGU

Advantages of CART

- Simple to understand, interpret, visualize.
- Variable Screening or Feature Selection.
  - Screening = Triagem
- It can handle both numerial and categorical data.
- Little effort for data preparation.

Disadvantages of CART

- Decision Tree Learners (DTL) can create complex trees that do not generalize data well.
  - Overfitting
- Decision Trees can be unstable because of small variations (variance) in the data.
  - Lowered by methods of bagging and boosting.
- Greedy algorithms cannot garantee to return the globally optimal decision tree.
  - Mitigated by training multiple trees.
    - Features and samples are randomly sampled with replacement.
- DTLs can create bias trees if some classes dominate.
  - Recommendation: balance the data set prior to fitting.

Growing a tree

- Features to choose
- Conditions for splitting
- Knowing when to stop
- Pruning
  - Prunning = Podar, Aparar

Regression trees

- The value obtained by terminal nodes in the training data is the mean response of observation falling in that region.
  - So we make predictions using the mean value.
- In classification trees, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region.

Stopping criteria

- This splitting process is continued until a user defined stopping criteria is reached.
- For example: we can tell the the algorithm to stop once the number of observations per node becomes less than 50.

Numeric data

Weight	Heart Disease
220	Yes
180	Yes
225	Yes
190	No
155	No

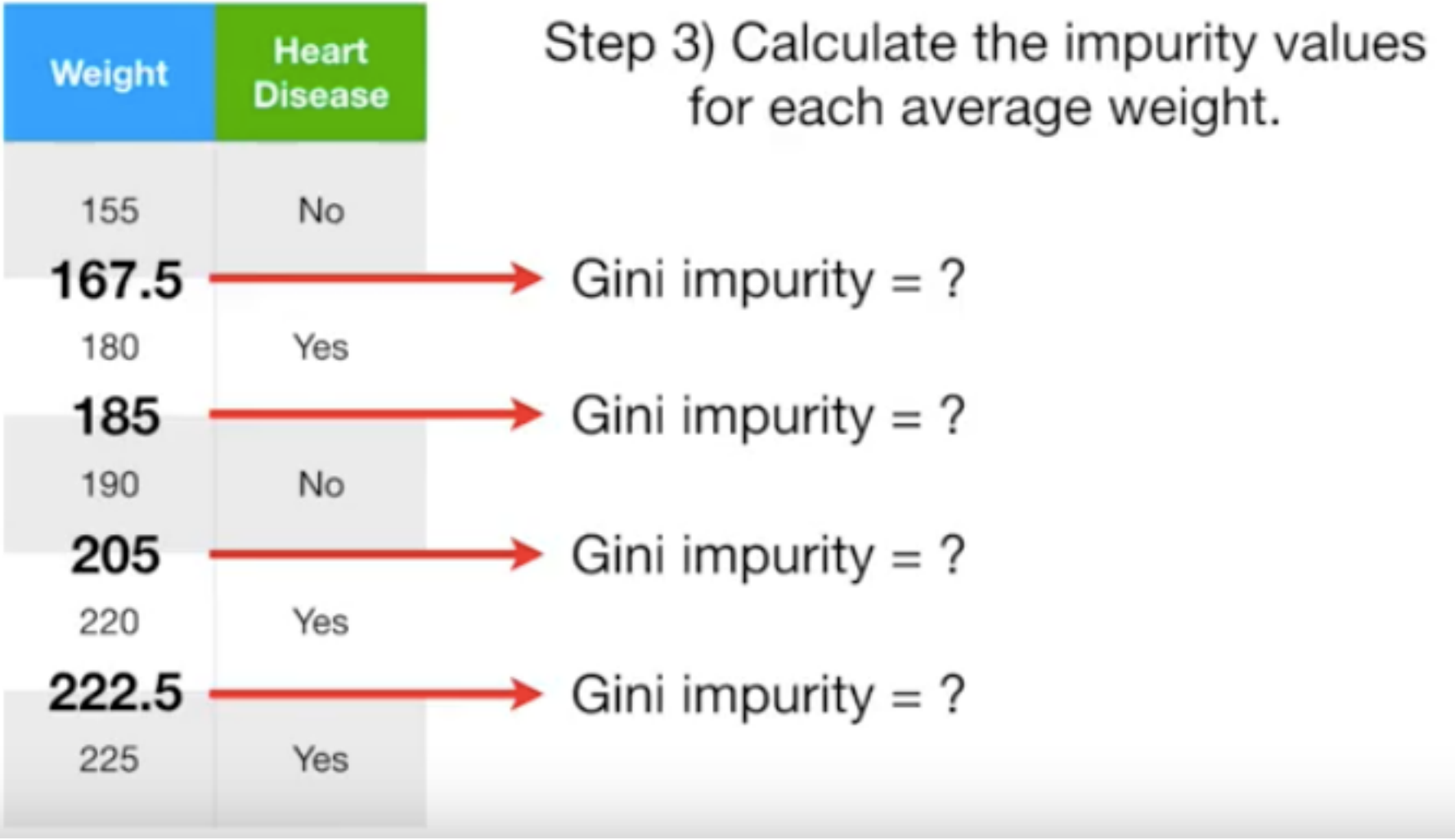
1) Sort the patients by weight, lowest to highest.

	Weight	Heart Disease
Lowest	155	No
	180	Yes
	190	No
	220	Yes
Highest	225	Yes

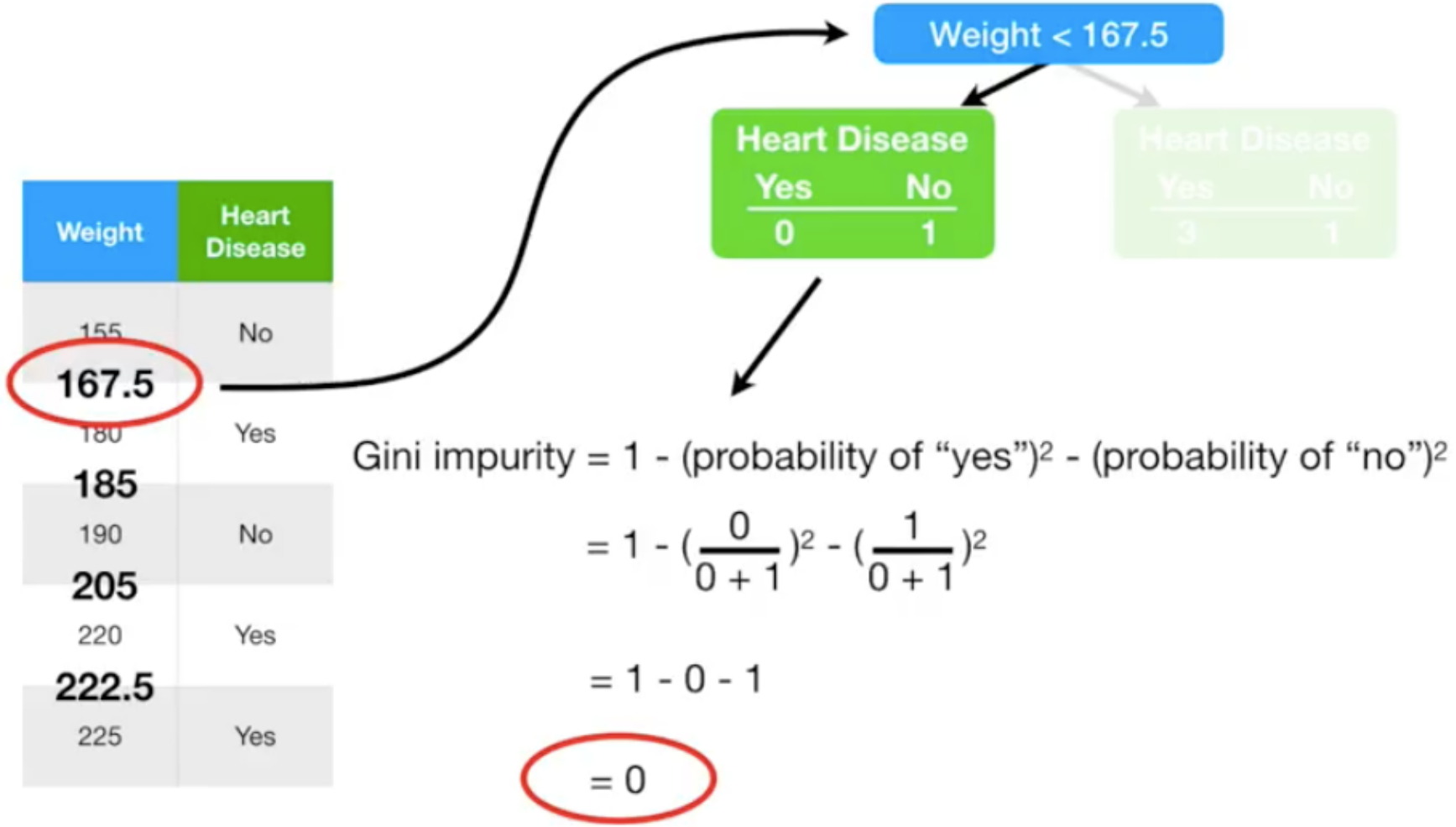
Numeric data

Step 2) Calculate the average weight for all adjacent patients.

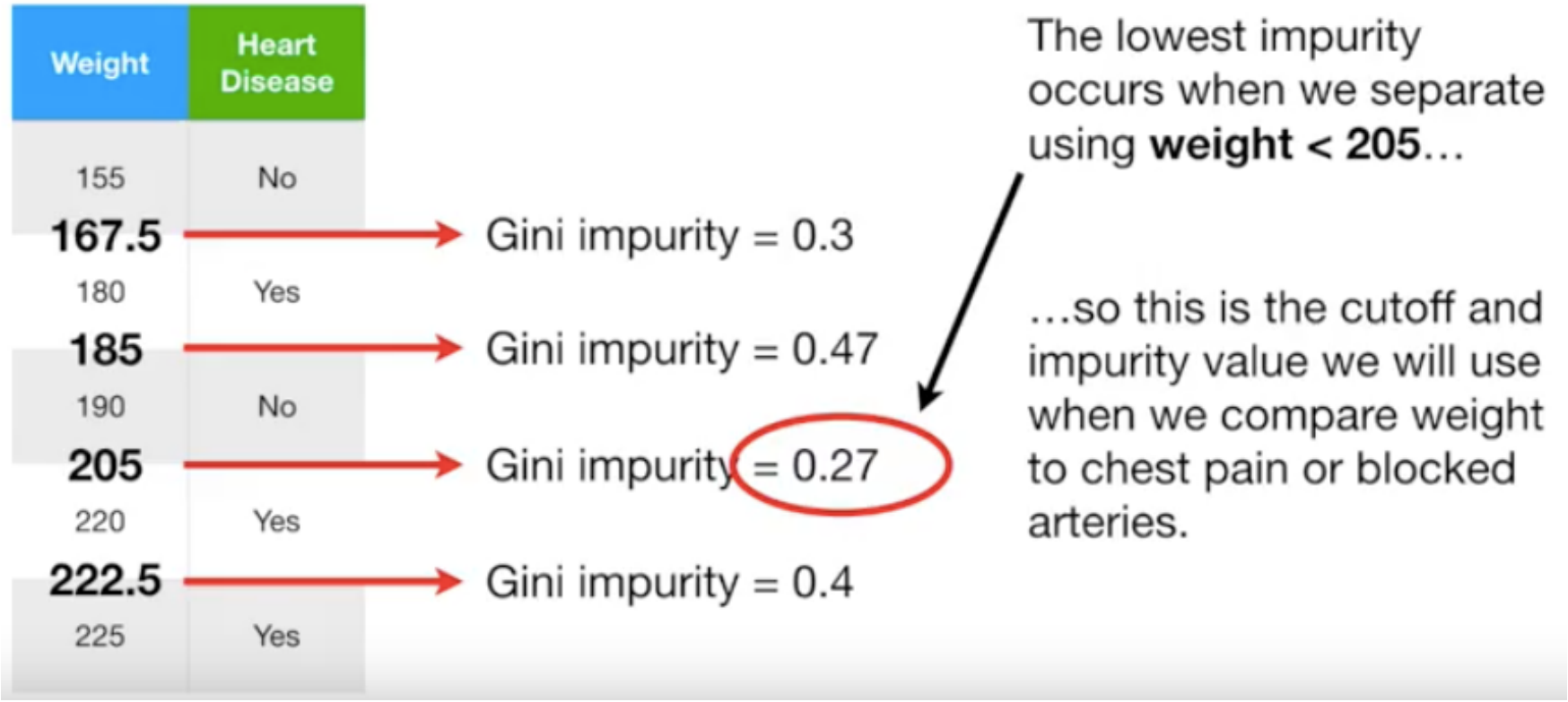
Numeric data



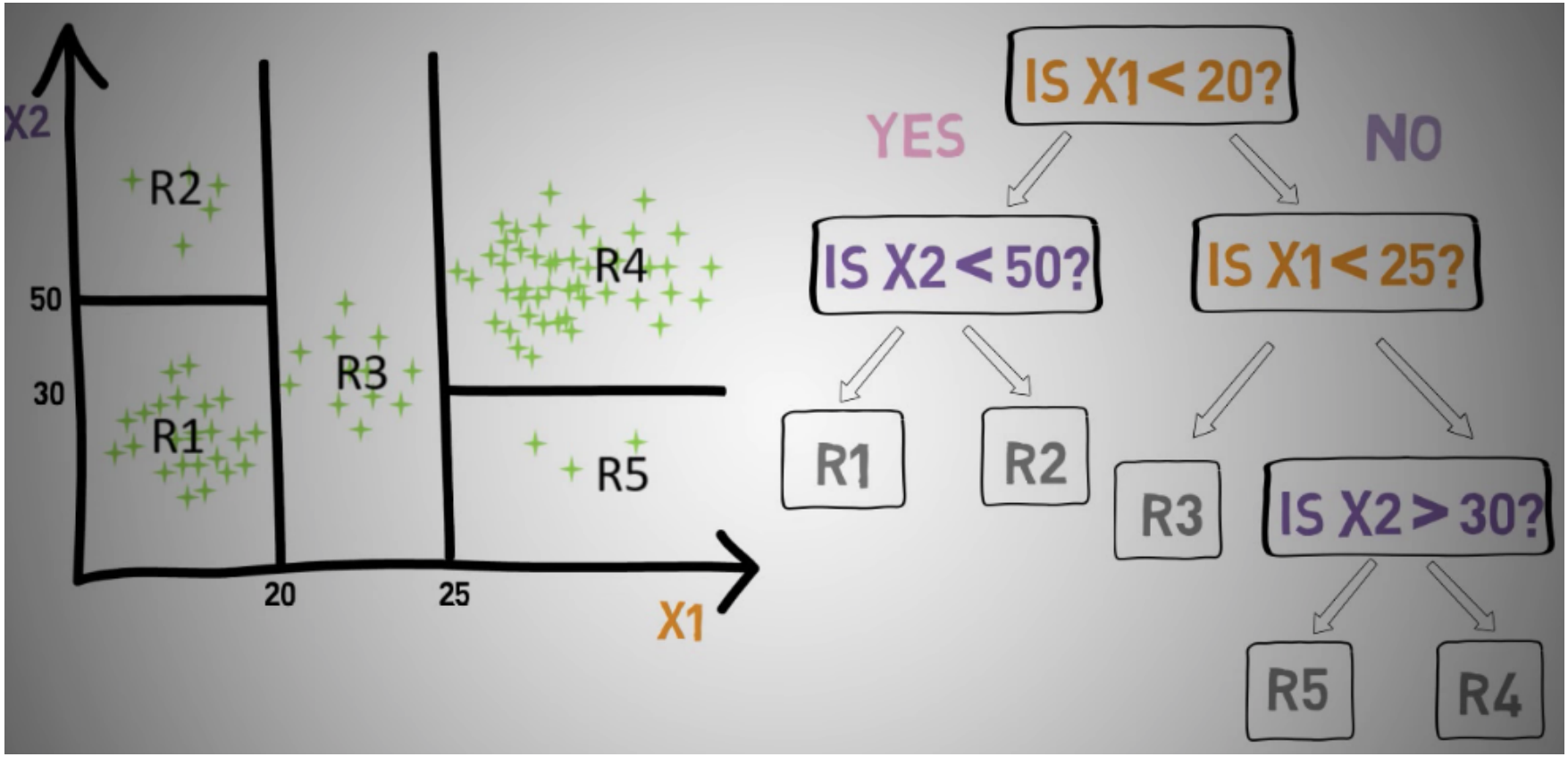
Numeric data



Numeric data



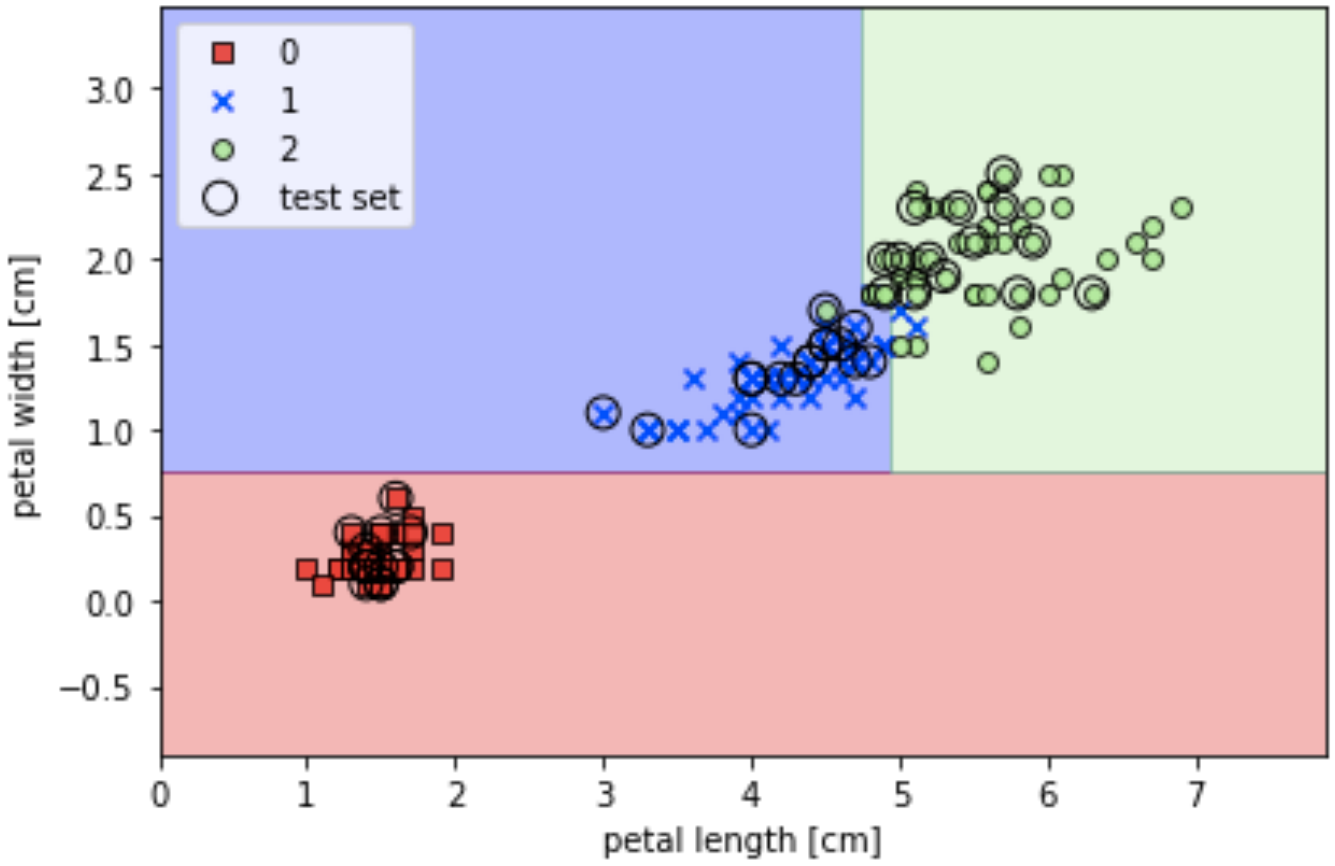
Building the tree



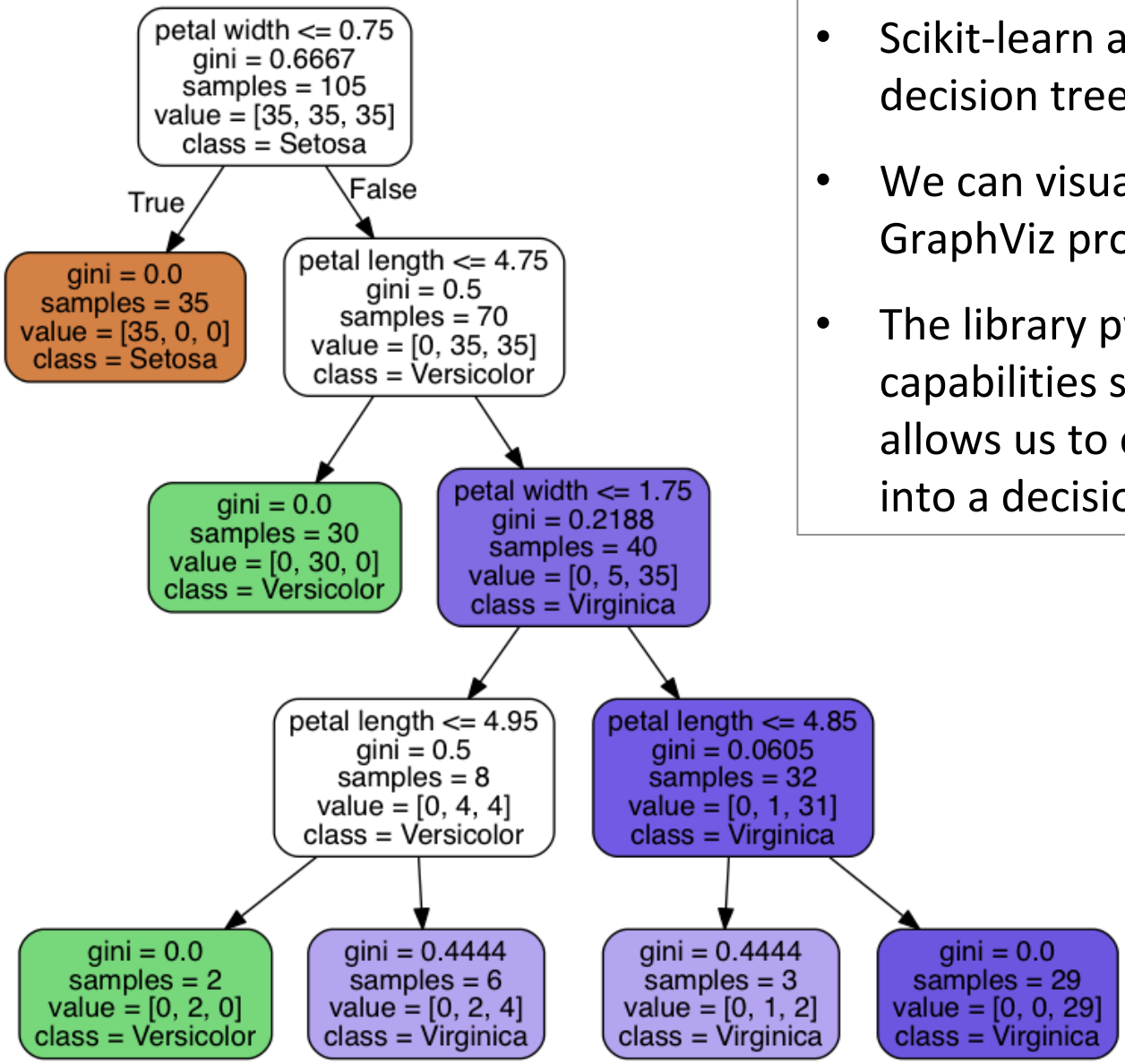
Building a decision tree

- Decision trees can build complex decision boundaries by dividing the feature space into rectangles.
- The deeper the decision tree, the more complex the decision boundary becomes, which can easily result in overfitting.
- Feature scaling may be desired for visualization purposes, but it is not a requirement for decision tree algorithms.

Building a decision tree



Building a decision tree



- Scikit-learn allows us to export the decision tree as a .dot file.
- We can visualize it using the GraphViz program.
- The library pydotplus has capabilities similar to GraphViz and allows us to convert .dot data files into a decision tree image file.

Pruning

- Top down approach (early stopping)
- Bottom up approach (error rate)

Pruning

1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

- Sklearn's decision tree classifier does not currently support pruning.
- Advanced packages like xgboost have adopted tree pruning in their implementation.

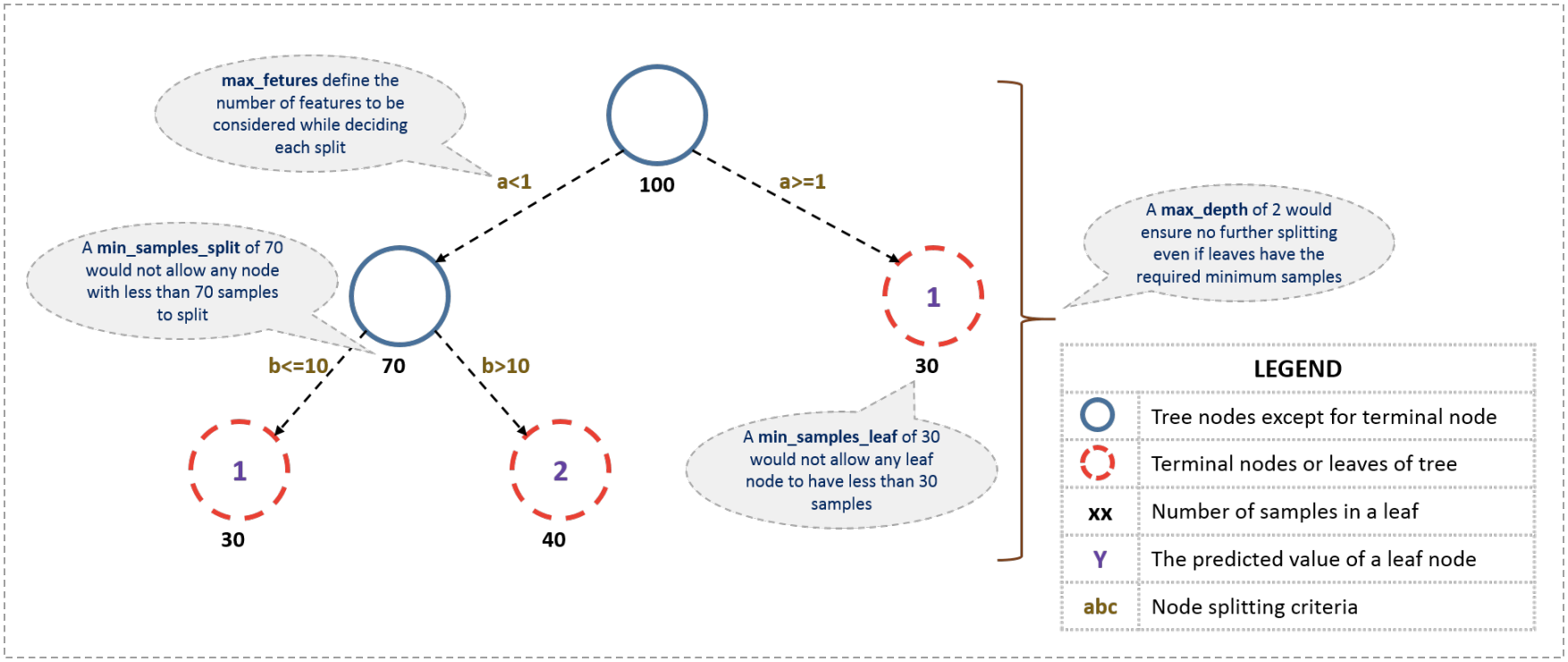
Pros

- Not every relation is linear. Non-linear relations are captured by DTs.
- Easy to understand and visualise.
- Can be used for feature engineering.
- No Assumptions - God Level.
- Very little data preparation needed for algorithm.
- Model validation using statistical test, influences model reliability.

Cons

- If not tuned well, may lead to overfitting.
- Non-parametric, no optimisation.
  - But, hyperparameter tuning.
  - So, once it gives limit you can not go further.
- Unstable.
  - Small variation in data -> completely different tree formation.
- In case of imbalanced dataset, decision trees are biased.
  - However, by using proper splitting criteria, this issue can be resolved.

Most Important Parameters



Most Important Parameters

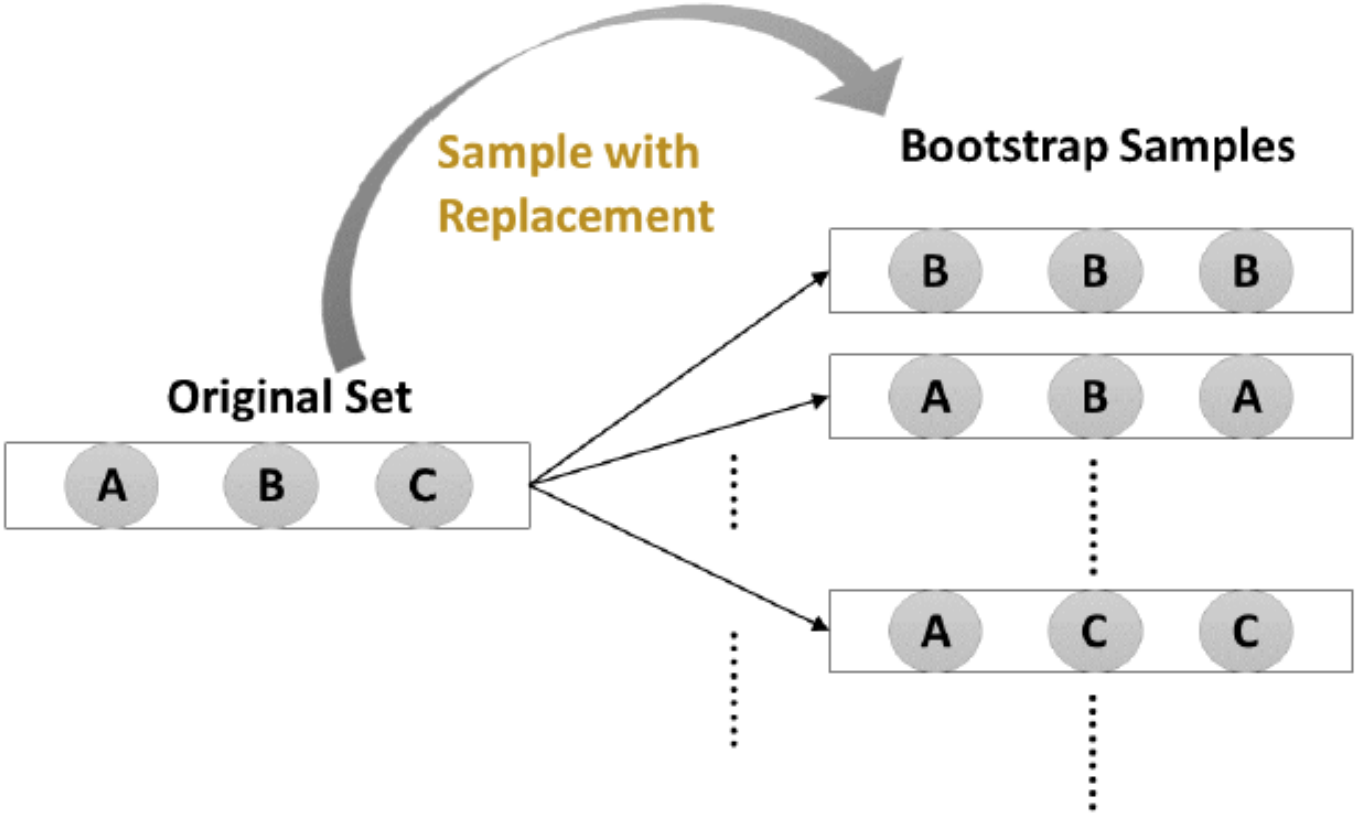
- **Maximum Features:** At each node, while splitting either we can chose best feature from pool of all the features or limited number of random features.
  - This parameter adds a little randomness - good generalised model.
- **Minimum Samples Split:** Minimum number of sample required to split a node. This parameter helps in reducing overfitting.
  - High value: Underfitting, Low value: Overfitting
- **Maximum Depth of a Tree:** Most influential parameter. Gives limit on vertical depth decide upto which level pruning is required.
  - Higher value: Overfitting, Lower value: Underfitting

Ensemble

- **Voting Classifier**
  - same training set
  - ≠ algorithms.

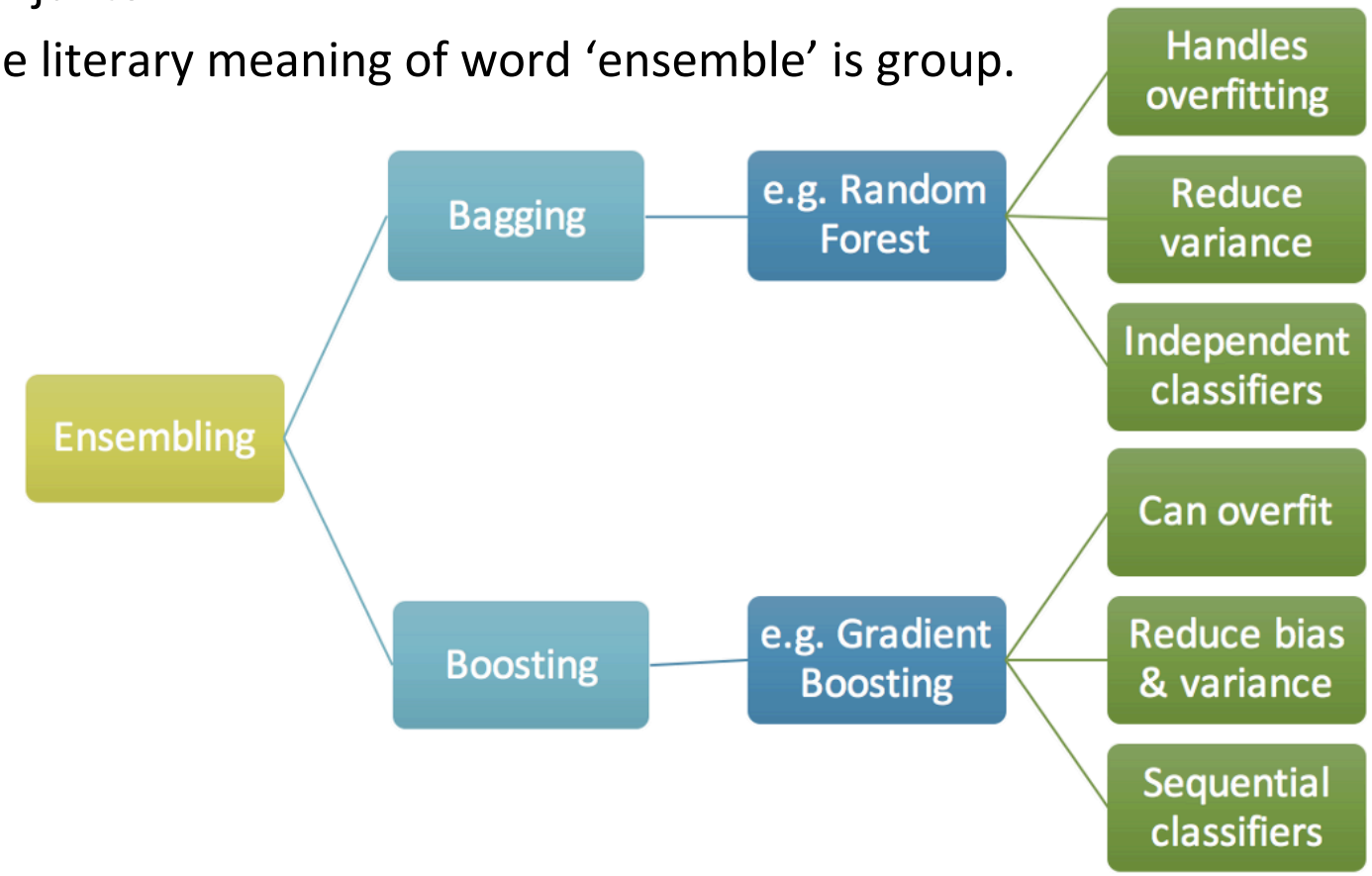
- **Bagging / Bootstrap Aggregation**
  - one algorithm,
  - ≠ subsets of the training set.

Bootstrap


Sample with
Replacement
Bootstrap Samples
Original Set
A
B
C
B
B
B
A
B
A
A
C
C

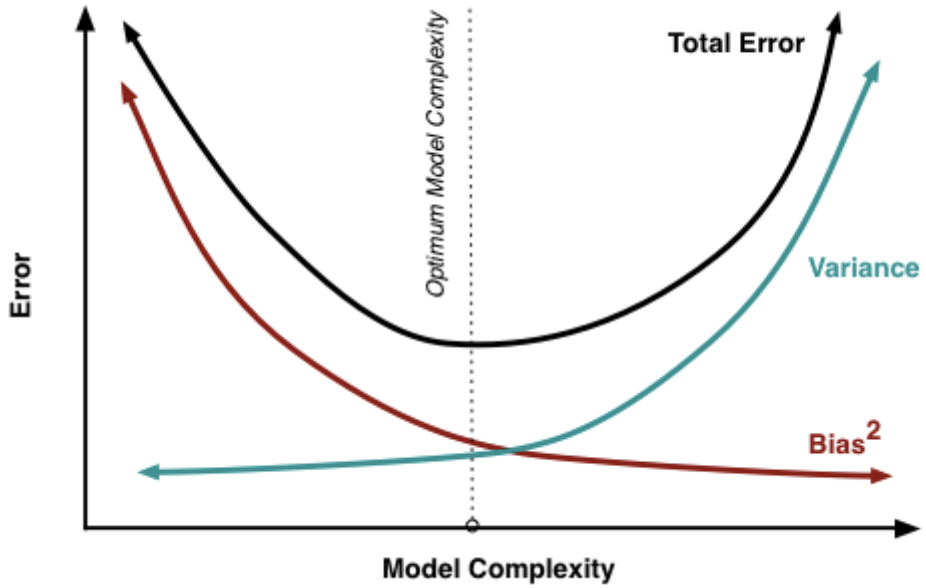
Ensemble

- Definition
  - A group of items viewed as a whole rather than individually.
  - Conjunto.
  - The literary meaning of word ‘ensemble’ is group.



Ensemble

- Like every other model, a tree based model also suffers from the plague of bias and variance.
- A champion model should maintain a balance between these two types of errors.
  - This is known as the trade-off management of bias-variance errors.
- Ensemble learning is one way to execute this trade off analysis.
- Commonly used ensemble methods include: Bagging, Boosting and Stacking.

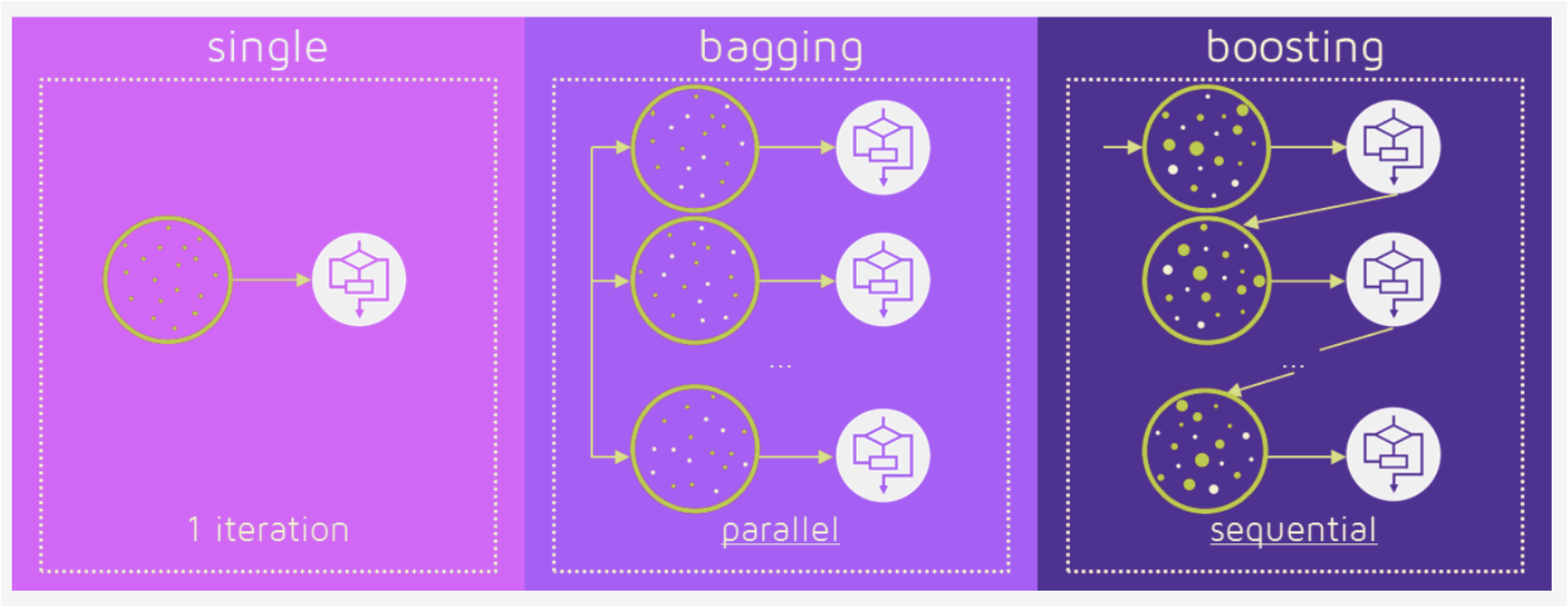


Ensemble

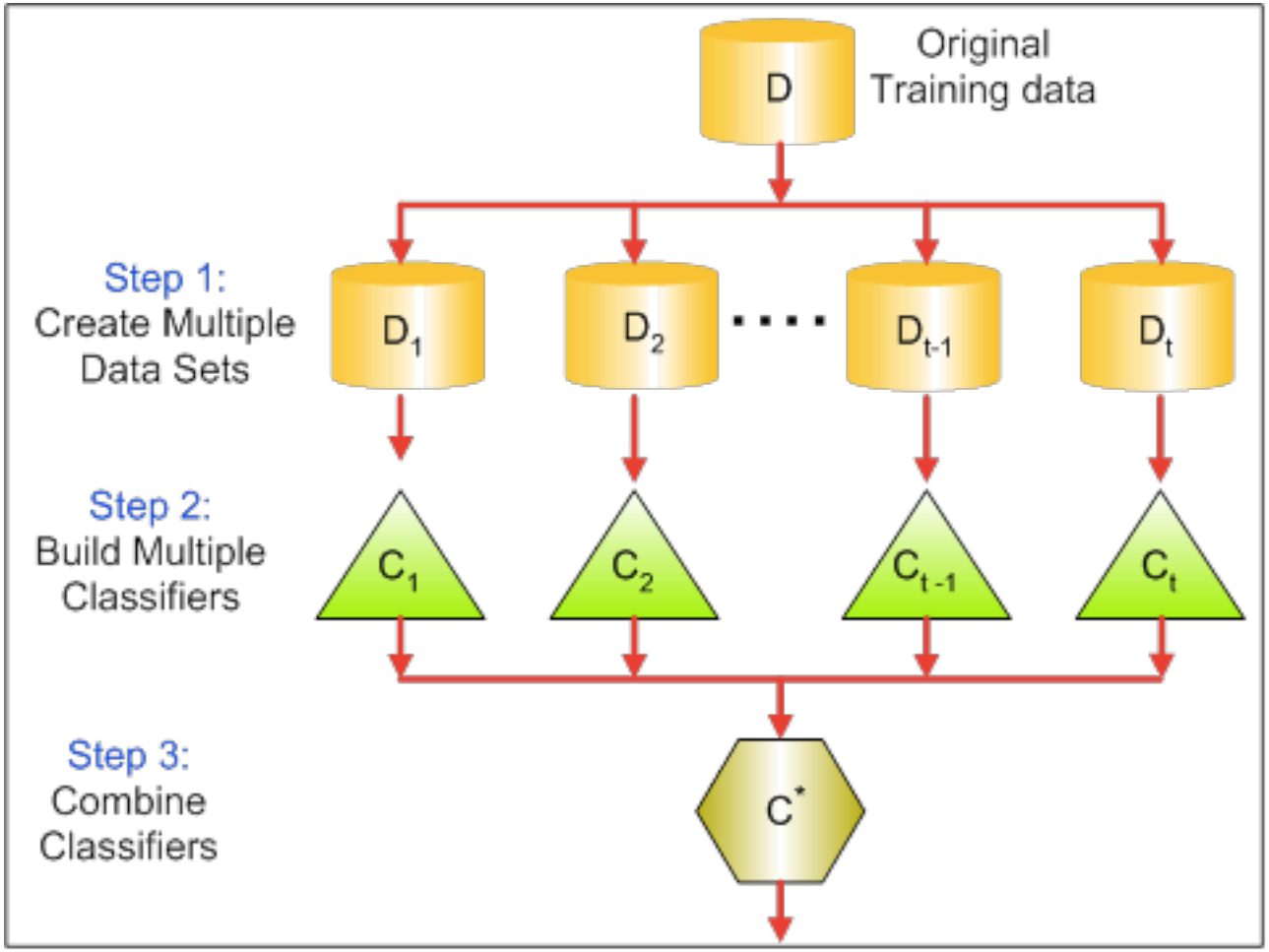
- Bagging is a simple ensembling technique in which we build many independent predictors/ models/learners and combine them using some model averaging techniques.
  - e.g. weighted average, majority vote or normal average

- Boosting is an ensemble technique in which the predictors are not made independently, but sequentially.

Ensemble



Bagging


D
Original
Training data
Step 1:
Create Multiple
Data Sets
D1
D2
Dt-1
Dt
Step 2:
Build Multiple
Classifiers
C1
C2
Ct-1
Ct
Step 3:
Combine
Classifiers
C*

Bagging

- Create Multiple DataSets
  - Sampling with replacement
  - New data sets can have a fraction of the columns as well as rows
- Build Multiple Classifiers
  - Classifiers are built on each data set
- Combine Classifiers
  - The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem.
- **The combined values are generally more robust than a single model.**

Random Forest

- Base estimator: Decision Tree
- Each estimator is trained on a different bootstrap sample having the same size as the training set.
- RF introduces further randomization in the training of individual trees.
- d features are sampled at each node **without replacement** (d < total number of features)
- scikit: n_features = sqrt(number of features)

Random Forest

- Capable of performing both regression and classification tasks.
- It performs dimensional reduction methods.
- It treats missing values, outlier values and other essential steps of data exploration.
- Type of ensemble learning method, where **a group of weak models combine to form a powerful model**.
- It captures the variance of several input variables at the same time and enables high number of observations to participate in the prediction.

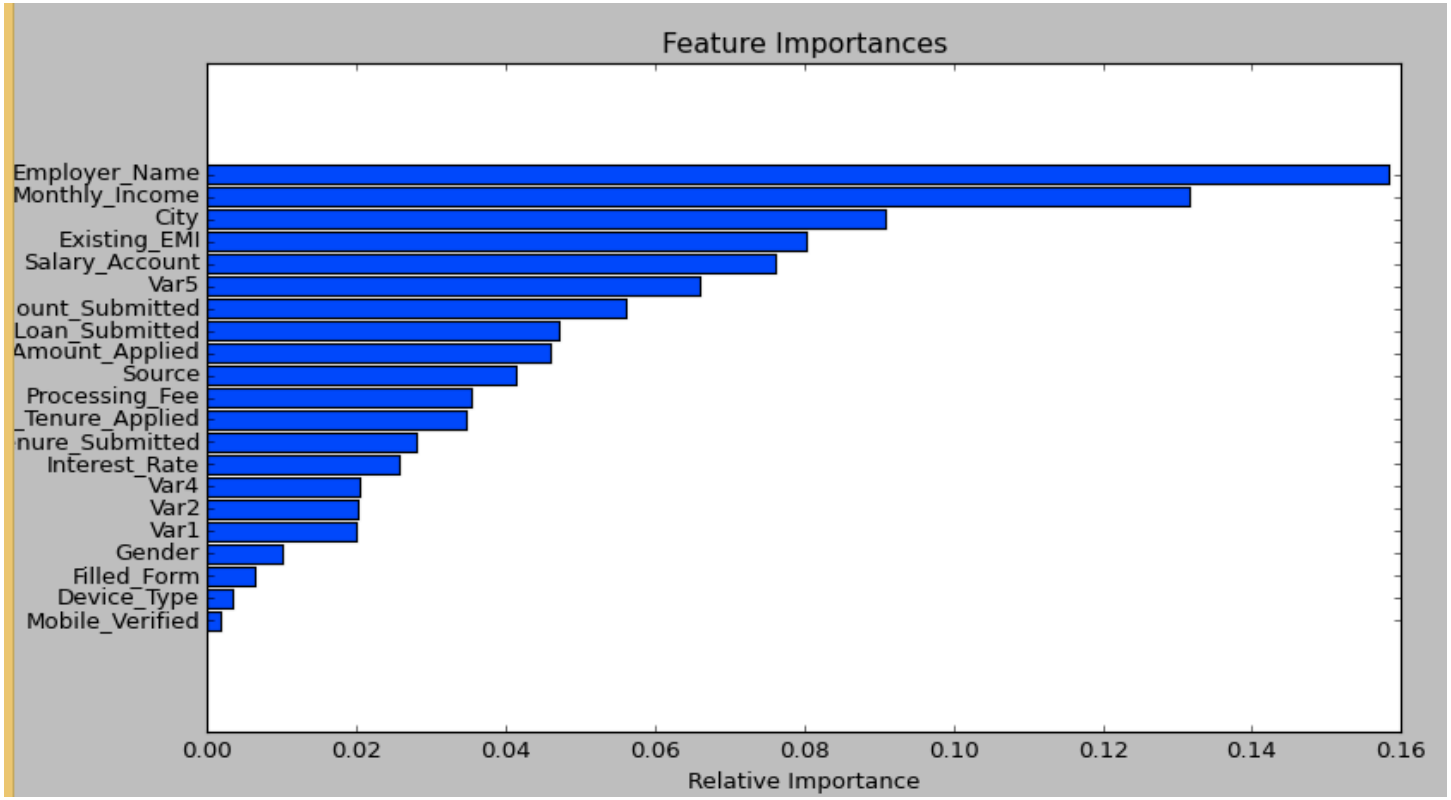
Random Forest

- Each tree gives a classification and we say the tree “votes” for that class.
- The forest chooses the classification having the most votes over all the trees in the forest.

- In case of regression, it takes the average of outputs by different trees.

Random Forest

- It handles large data set with higher dimensionality.
- It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.



Random Forest

- Good classification performance, scalability, and ease of use.
- Considered as an ensemble of decision trees.
- Idea:
  - Average multiple (deep) decision trees that individually suffer from high variance, to build a more robust model that has a better generalization performance and is less susceptible to overfitting.

Random Forest

1. Draw a random bootstrap sample of size n (randomly choose n samples from the training set with replacement).
2. Grow a decision tree from the bootstrap sample. At each node:
    1. Randomly select d features without replacement.
    2. Split the node using the feature that provides the best split according to the objective function, for instance, maximizing the information gain.
3. Repeat the steps 1-2 k times.
4. Aggregate the prediction by each tree to assign the class label by majority vote.

Random Forest

- Cons
  - They don't offer the same level of interpretability as decision trees.
- Pros
  - We don't have to worry so much about choosing good hyperparameter values.
  - We typically don't need to prune the random forest since the ensemble model is quite robust to noise from the individual decision trees.

RF Parameters

- The only parameter that we really need to care about in practice is the number of trees that we choose for the random forest.
  - Typically, the larger the number of trees, the better the performance of the random forest classifier at the expense of an increased computational cost.
- Size n of the bootstrap sample (step 1)
  - Via the sample size n of the bootstrap sample, we control the bias-variance tradeoff of the random forest.
  - Decreasing the size n of the bootstrap sample increases the diversity among the individual trees, since the probability that a particular training sample is included in the bootstrap sample is lower.
  - Shrinking the size of the bootstrap samples may increase the randomness of the random forest, and it can help to reduce the effect of overfitting.
- The number of features d that is randomly chosen for each split (step 2.1).

RF Parameters

- In most implementations the size of the bootstrap sample is chosen to be equal to the number of samples in the original training set.
  - This usually provides a good bias-variance tradeoff.
- For the number of features d at each split, we want to choose a value that is smaller than the total number of features in the training set.
  - A reasonable default that is used is d = sqrt(m) , where m is the number of features in the training set.

Boosting

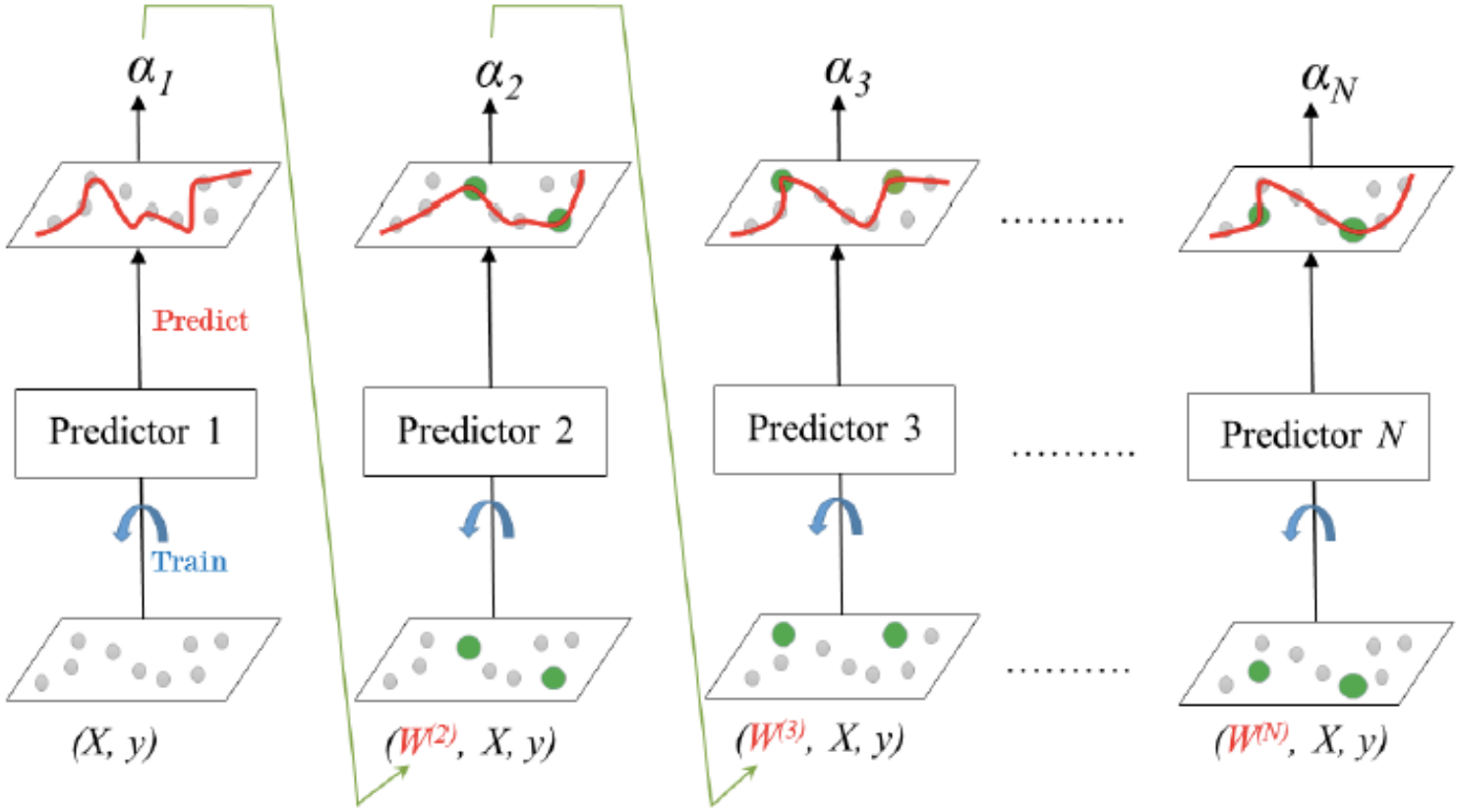
- **Boosting**: Ensemble method combining several weak learners to form a strong learner.

- **Weak learner**: Model doing slightly better than random guessing.

- Example of weak learner: Decision stump (CART whose maximum depth is 1).

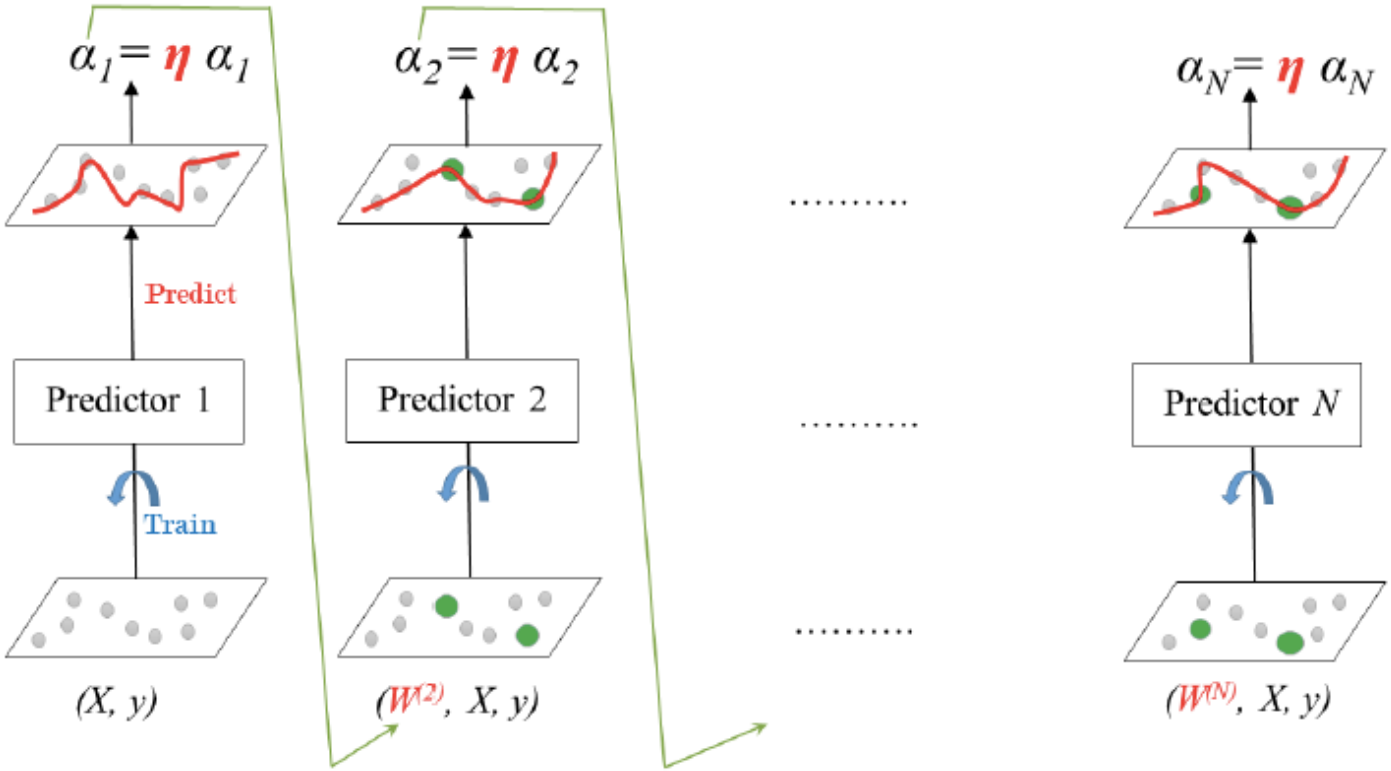
- Train an ensemble of predictors sequentially.
- Each predictor tries to correct its predecessor.

AdaBoost

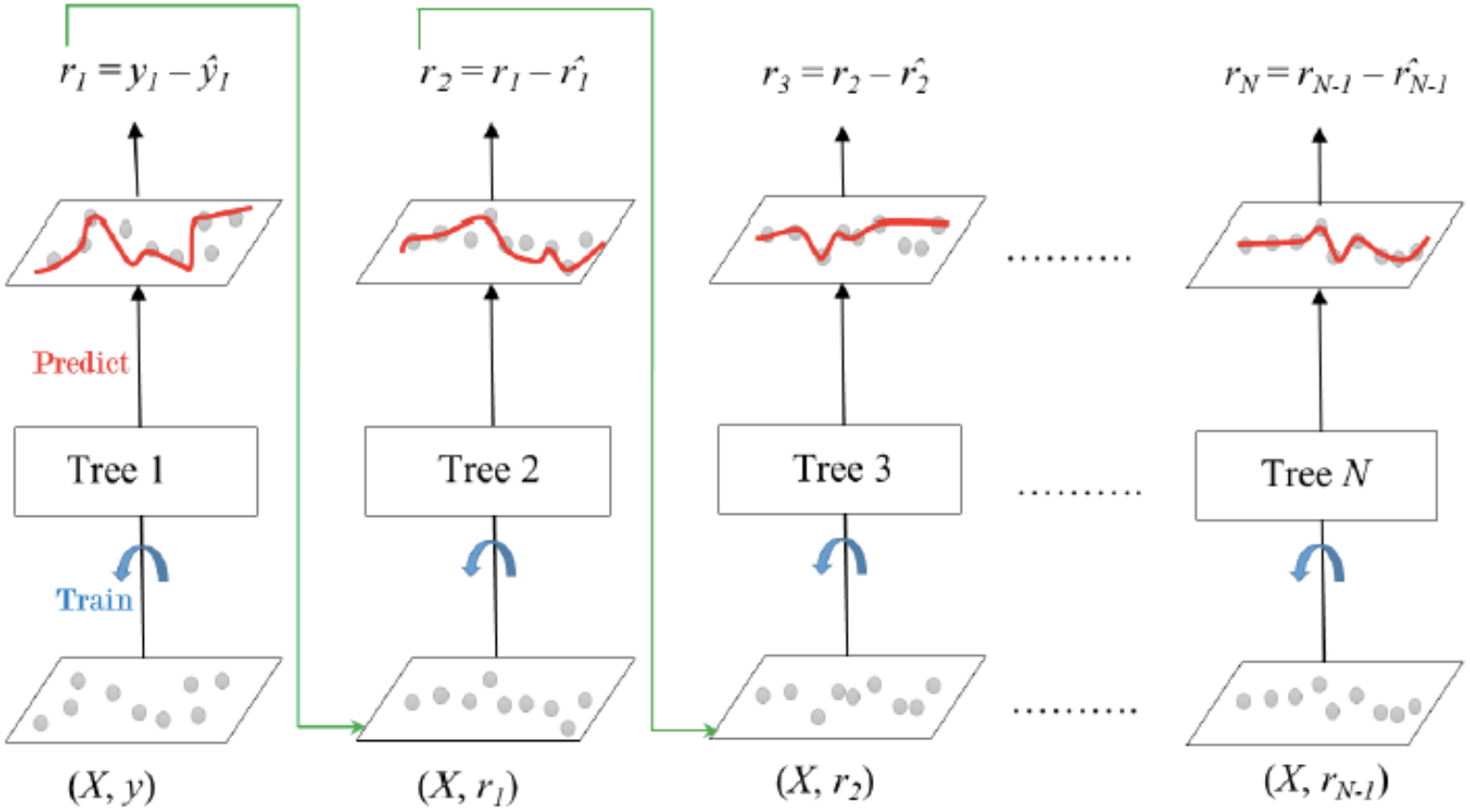

α_1
α_2
α_3
α_N
Predict
Predictor 1
Predictor 2
Predictor 3
Predictor N
Train
(X, y)
(W(2), X, y)
(W(3), X, y)
(W(N), X, y)

AdaBoost

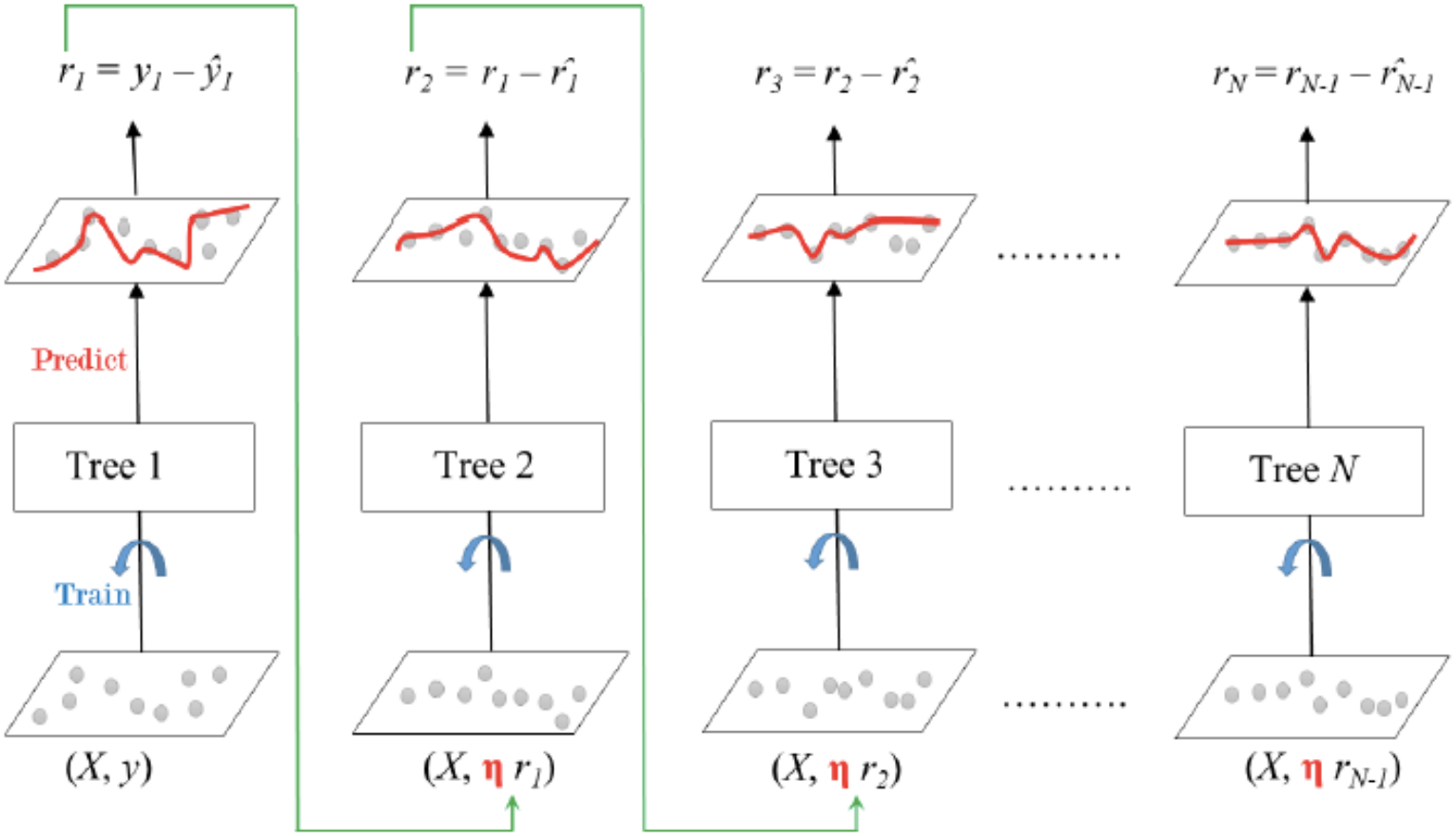
Learning rate: $0 < \eta \leq 1$



Gradient Boosted Trees



Gradient Boosted Trees - Shrinkage



Stochastic Gradient Boosting

- Each tree is trained on a random subset of rows of the training data.
- The sampled instances (40%-80% of the training set) are sampled without replacement.
- Features are sampled (without replacement) when choosing split points. Result: further ensemble diversity.
- Effect: adding further variance to the ensemble of trees.

Gradient Boosting Parallelism

- Implemented in the construction of individual trees, rather than in creating trees in parallel like random forest.
- In the efficient preparation of the input data to aid in the speed up in the construction of trees.
- In boosting, trees are added to the model sequentially.

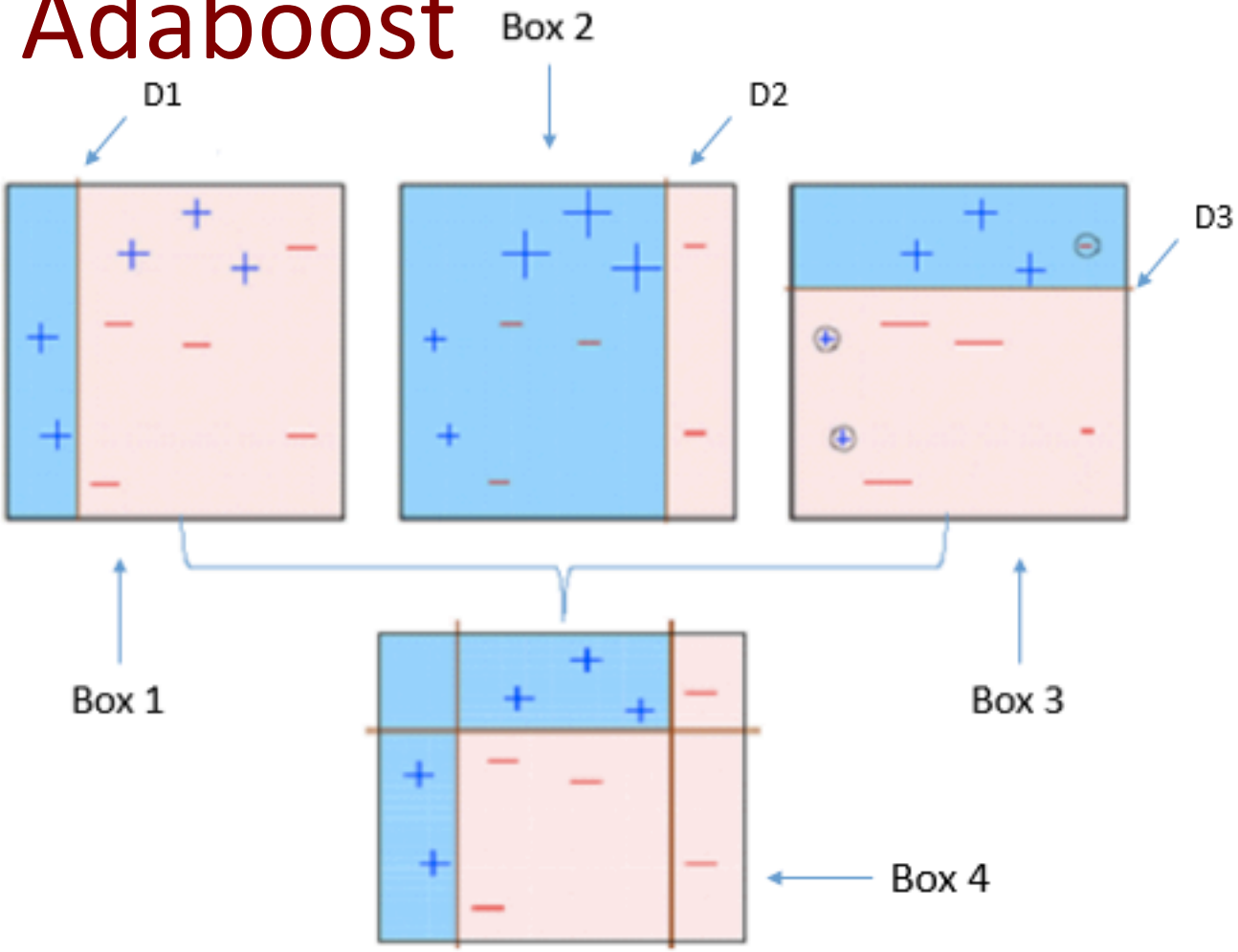
Boosting

- Boost = Impulsionar, Encorajar
- ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.
- Weak learner: rules individually are not powerful enough.
- To convert weak learner to strong learner, we’ll combine the prediction of each weak learner using methods like:
  - Using average/ weighted average
  - Considering prediction has higher vote

Additive Model

- Weak learner sub-models or more specifically decision trees
  - After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient).
  - We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss).
- This approach is called functional gradient descent or gradient descent with functions.
- The output for the new tree is then added to the output of the existing sequence of trees in an effort to correct or improve the final output of the model.

Adaboost



- We have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner.
- You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

Adaboost

- **Ada**ptive **Boost**ing
- It fits a sequence of weak learners on different weighted training data.
- It starts by predicting original data set and gives equal weight to each observation.
- If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly.
- Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.

Adaboost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
clf = AdaBoostClassifier(n_estimators=100,
base_estimator=dt,learning_rate=1)

#Above I have used decision tree as a base estimator, you
can use any ML learner as base estimator if it accepts
sample weight
clf.fit(x_train,y_train)
```

Adaboost − main parameters

- **n_estimators**
  - It controls the number of weak learners.
- **learning_rate**
  - Controls the contribution of weak learners in the final combination. There is a trade-off between learning_rate and n_estimators.
- **base_estimators**
  - It helps to specify different ML algorithm.

How boosting identify weak rules?

- We apply ML algorithms with a different distribution.
- Each time base learning algorithm is applied, it generates a new weak prediction rule.
- After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

How do we choose different distribution for each round?

1. The base learner takes all the distributions and assign equal weight or attention to each observation.
2. If there is any prediction error caused by first base learning algorithm, then **we pay higher attention to observations having prediction error**. Then, we apply the next base learning algorithm.
3. Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.
4. Combine the outputs from weak learner and create a strong learner which eventually improves the prediction power of the model.

Boosting pays higher **focus on examples which are mis-classified or have higher errors** by preceding weak rules.

Gradient Boosting

- It trains many models sequentially.
- Each new model gradually minimizes the loss function (y = ax + b + e, e needs special attention as it is an error term) of the whole system using Gradient Descent method.
- The learning procedure consecutively fit new models to provide a more accurate estimate of the response variable.

Gradient Boosting

- Elements:
  - A loss function to be optimized.
  - A weak learner to make predictions.
  - An additive model to add weak learners to minimize the loss function.

Subsamples

- at each iteration a subsample of the training data is drawn at random (without replacement) from the full training dataset.
- The randomly selected subsample is then used, instead of the full sample, to fit the base learner.
- Variants of stochastic boosting:
  - Subsample rows before creating each tree.
  - Subsample columns before creating each tree
  - Subsample columns before considering each split.

Stochastic Gradient Boosting

- Enhancements to basic gradient boosting
  - Tree Constraints
  - Shrinkage
  - Random sampling
  - Penalized Learning

GBM algorithm for 2 classes

1. Initialize the outcome
2. Iterate from 1 to total number of trees
   2.1 Update the weights for targets based on previous run (higher for the ones mis-classified)
   2.2 Fit the model on selected subsample of data
   2.3 Make predictions on the full set of observations
   2.4 Update the output with current results taking into account the learning rate
3. Return the final output.

Gradient Boosting

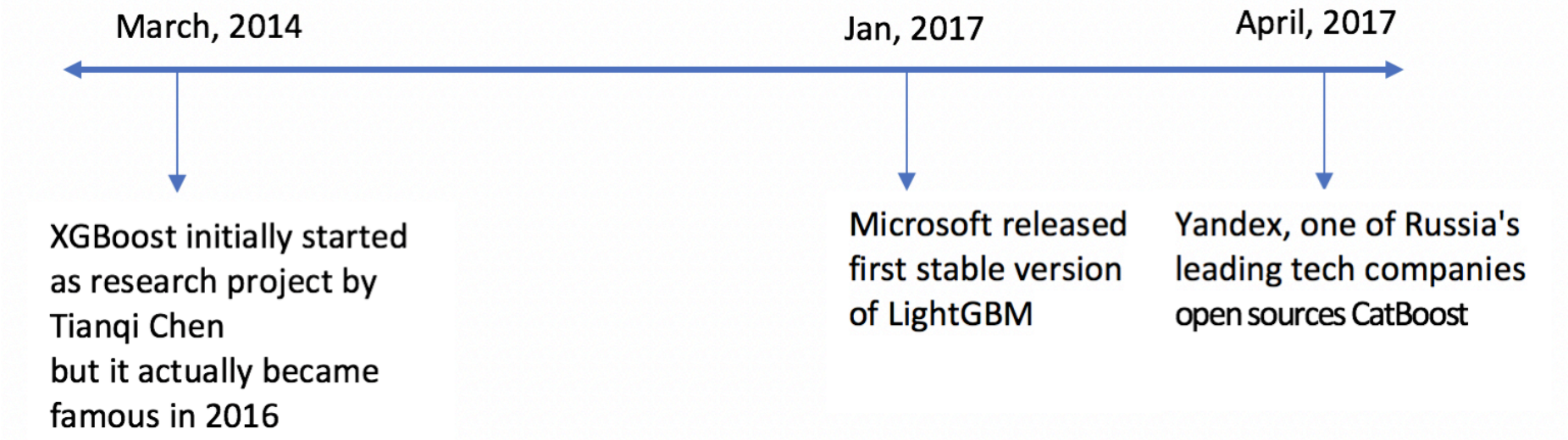
- **XGBoost**
  - de-facto algorithm for winning competitions at Analytics Vidhya and Kaggle, because it is extremely powerful.
  - But given lots and lots of data, even XGBOOST takes a long time to train.
- **LightGBM (Microsoft)**
  - Fast, distributed, high-performance gradient boosting framework.
  - Used for ranking, classification and many other machine learning tasks.
- **CatBoost**
  - “Category” and “Boosting”
  - It works well with multiple Categories of data, such as audio, text, image including historical data.
  - It yields state-of-the-art results without extensive data training typically required by other machine learning methods, and
  - Provides powerful out-of-the-box support for the more descriptive data formats that accompany many business problems.

Main tuning parameters

- learning_rate
  - It determines the impact of each tree on the final outcome.
- n_estimators
  - The number of sequential trees to be modeled.
- subsample
  - The fraction of observations to be selected for each tree.
  - Selection is done by random sampling.
  - Values slightly less than 1 make the model robust by reducing the variance.
  - Typical values ~0.8 generally work fine.

Gradient Boosting

- **XGBoost**
- **LightGBM (Microsoft)**
  - Fast, distributed, high-performance gradient boosting framework.
- **CatBoost**
  - “Category” and “Boosting”
  - It works well with multiple Categories of data, such as audio, text, image including historical data.



XGBoost

- Regularization
- Parallel Processing
- High Flexibility
  - It allows users to define custom optimization objectives and evaluation criteria.
- Handling Missing Values
- Tree Pruning
  - GBM would stop splitting a node when it encounters a negative loss in the split.
  - XGBoost on the other hand make splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

XGBoost

- Built-in Cross-Validation
  - It allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
- Continue on Existing Model
  - User can start training an XGBoost model from its last iteration of previous run.
  - Scikit-Learn GBM also has this feature.

Regularization

- shrinkage or eta or learning rate.
- Regularization means "way to avoid overfitting", so it is clear that the number of iterations M is crucial in that respect (a M that is too high leads to overfitting).
- The learning rate parameter ($v \in [0,1]$) in Gradient Boosting shrinks the contribution of each new base model -typically a shallow tree- that is added in the series.
- Similar to a learning rate in stochastic optimization, shrinkage reduces the influence of each individual tree and leaves space for future trees to improve the model.

Regularization

- Objective functions must always contain two parts: training loss and regularization.

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

- where L is the training loss function, and Ω is the regularization term.

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

Regularization

- Classical decision trees like CART are not used as weak learners, instead a modified form called a regression tree is used that has numeric values in the leaf nodes (also called terminal nodes).
- The values in the leaves of the trees can be called weights in some literature.
- The leaf weight values of the trees can be regularized using popular regularization functions, such as:
  - L1 regularization of weights.
  - L2 regularization of weights.

LightGBM vs XGBoost

- Adult dataset
  - http://archive.ics.uci.edu/ml/datasets/Adult

	accuracy score	auc score	execution time
LightGBM	0.861501	0.764492	00:00:00.283759
xgboost	0.861398	0.764284	00:00:02.047220

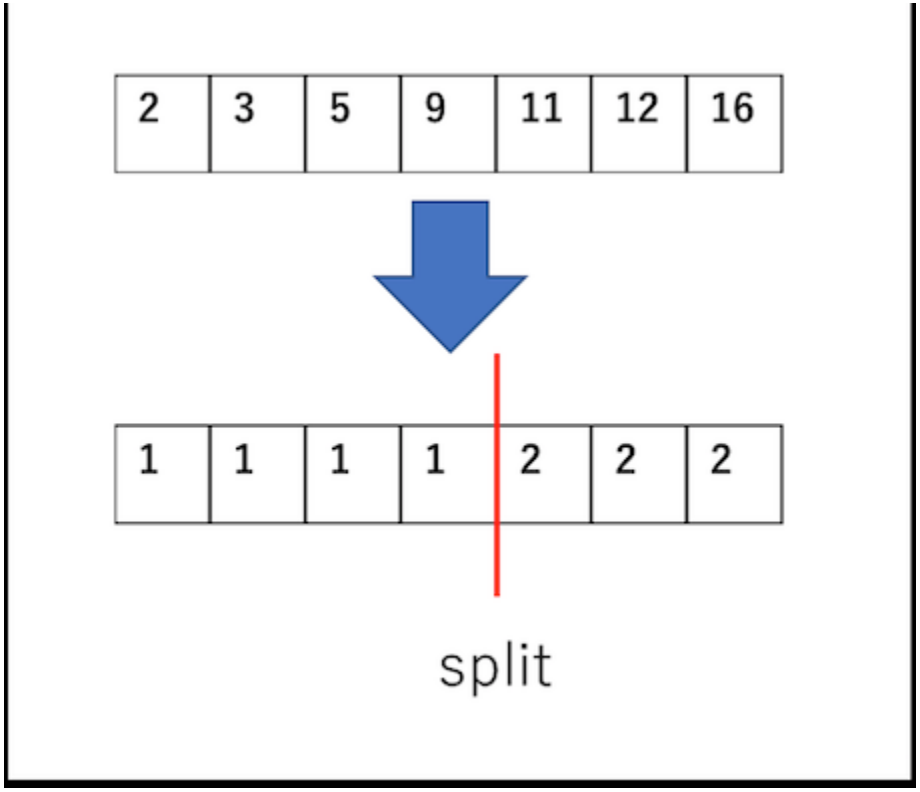
XGBoost also has a leave wise algorithm: use ‘hist’ as the value for ‘tree_method’ parameter. Execution speed is then comparable to that of lightgbm.

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

Spliting algorithms

- Pre-sorted algorithm
  - enumerates all possible split points on pre-sorted values.
  - highly inefficient in terms of computation power and memory .
- Histogram based algorithm
  - buckets continuous features into discrete bins to construct feature histograms during training.

Binning: reduce the number of candidate splits



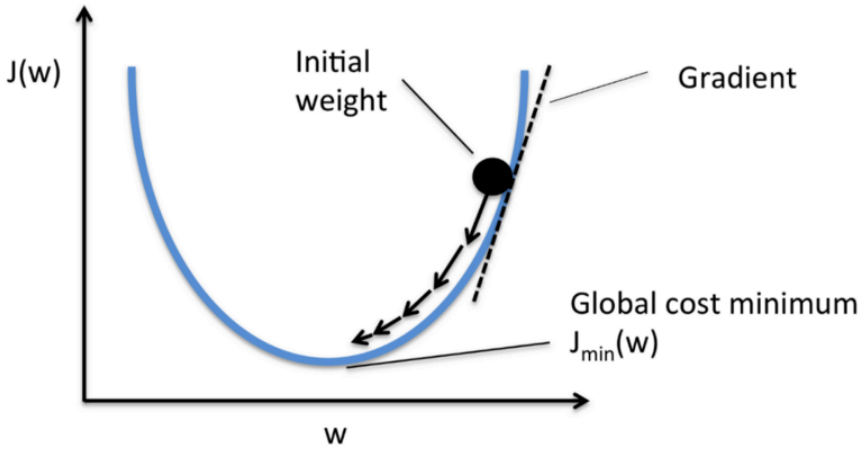
LightGBM and XGBoost utilise histogram based split finding (tree_method=hist).

Sklearn uses Pre-sorted algorithm (slow).

LightGBM

LightGBM aims to reduce complexity of histogram building by down sampling data and feature using GOSS and EFB.

Gradient



Gradient represents the slope of the tangent of the loss function, so logically if gradient of data points are large in some sense, these points are important for finding the optimal split point as they have higher error.

LightGBM from Microsoft

- Gradient-based One-Side Sampling (GOSS)
  - GOSS keeps all the instances with large gradients and performs random sampling on the instances with small gradients to estimate the information gain.
  - obtain quite accurate estimation of the information gain with a much smaller data size.
- Exclusive Feature Bundling (EFB)
  - bundle mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features. e.g. one-hot features.

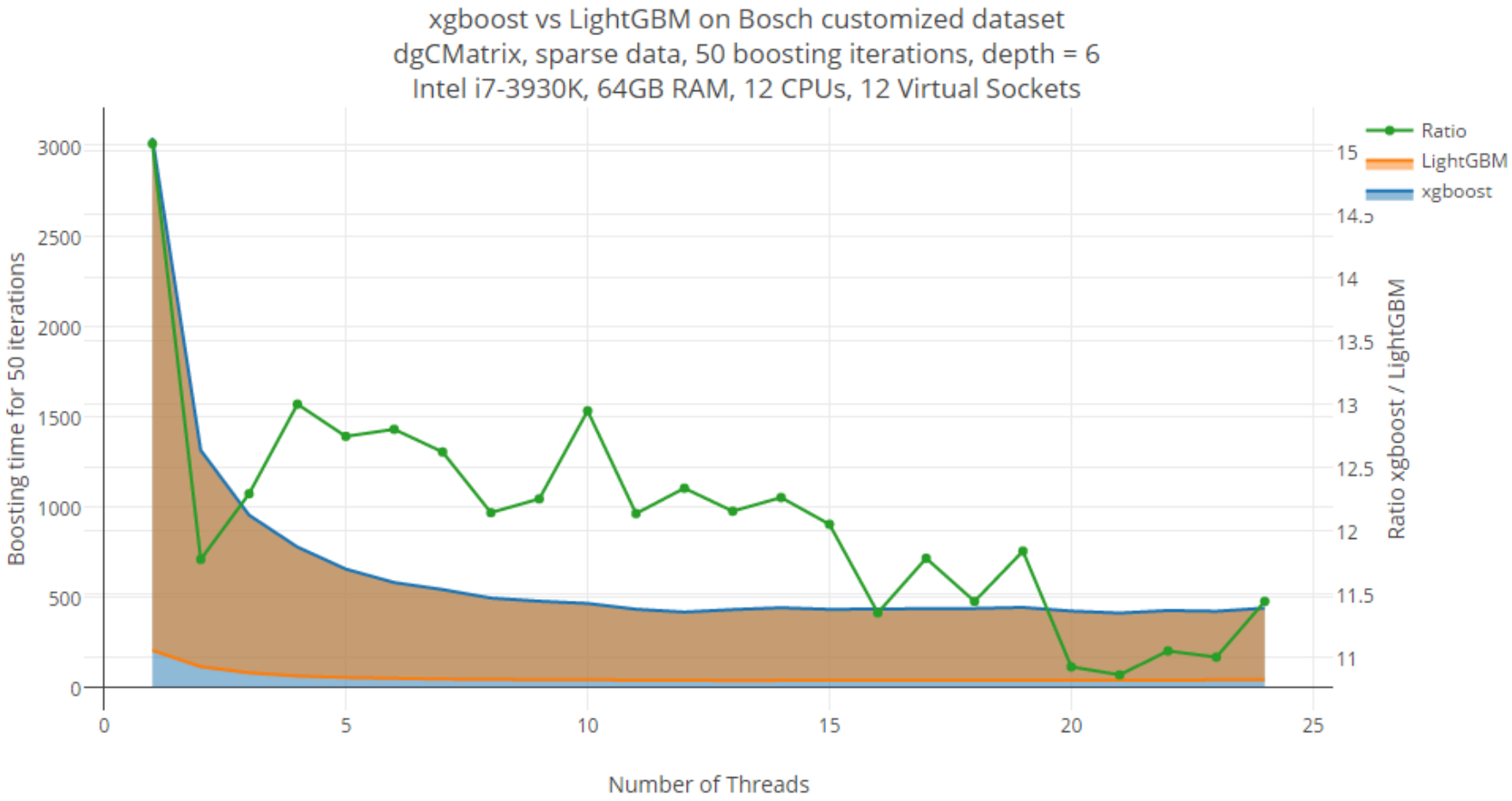
Merging features

feature1	feature2	feature_bundle
0	2	6
0	1	5
0	2	6
1	0	1
2	0	2
3	0	3
4	0	4

exclusive features reside in different bins

LightGBM from Microsoft

- It uses histogram based algorithm i.e it buckets continuous feature values into discrete bins which **fasten the training procedure**.
- Replaces continuous values to discrete bins which result in **lower memory usage**.
- It is capable of performing equally **good with large datasets** with a significant **reduction in training time** as compared to XGBOOST.
- **Parallel learning** supported.



LightGBM (binning) is between 11x to 15x faster than xgboost (without binning).

Log-loss value for test data

Lower is better

	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	**0.26974**	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	**0.13772**	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
Click prediction	**0.39090**	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
KDD appetency	0.07151	**0.07138** −0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
KDD churn	**0.23129**	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
KDD internet	**0.20875**	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%

https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/

Label Encoding vs One Hot Encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Categorial Features

Function	XGBoost	CatBoost	Light GBM
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model

CatBoost - categorical columns can be encoded as one-hot encoding.

If you don't pass any anything in **cat_features** argument, CatBoost will treat all the columns as numerical variables.

It is common to represent categorical features with one-hot encoding, but this approach is suboptimal for tree learners. Particularly for high-cardinality categorical features, a tree built on one-hot features tends to be unbalanced and needs to grow very deep to achieve good accuracy.

Kaggle Dataset of flight delays for the year 2015 as it has both categorical and numerical features.

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features	Passing indices of categorical features	Without passing indices of categorical features	Passing indices of categorical features
		0.992	0.999	0.842	0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

Referências

- RASCHKA, Sebastian; MIRJALILI, Vahid. Python Machine Learning, 2nd Ed. Packt Publishing, 2017.
- Datacamp - https://www.datacamp.com