Week 8: Ensemble learning

Introduction

You will also learn ensemble learning as a method of optimisation and regularisation techniques to avoid overfitting of models.

Recommended reading

Before reading this week's slides you should read the following chapters from Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 07 Ensemble Learning and Random Forests (all sections)

In the following slides, we summarise some of the important points discussed in the above chapters. We will then focus on the Ed activities based on the content discussed in the chapters.

Boosting

A learning algorithm is **weak** if the accuracy of the algorithm is only slightly better than random guessing. On the other hand, a learning algorithm is **strong** if the accuracy of the algorithm is very high.

Boosting is an ensemble method that converts a set of weak learners to a strong learner. Boosting algorithms use an iterative framework, where weak learners are sequentially trained and added to the framework. Each new weak learner tries to correct its predecessor, in order to improve overall accuracy. We will discuss the most popular boosting methods: **AdaBoost** (Adaptive Boosting), **Gradient Boosting**, and **XGBoost**.

Below we get the overview of boosting from a simple diagram where we see how the dataset is reconstructed at different stages of the boosting process.

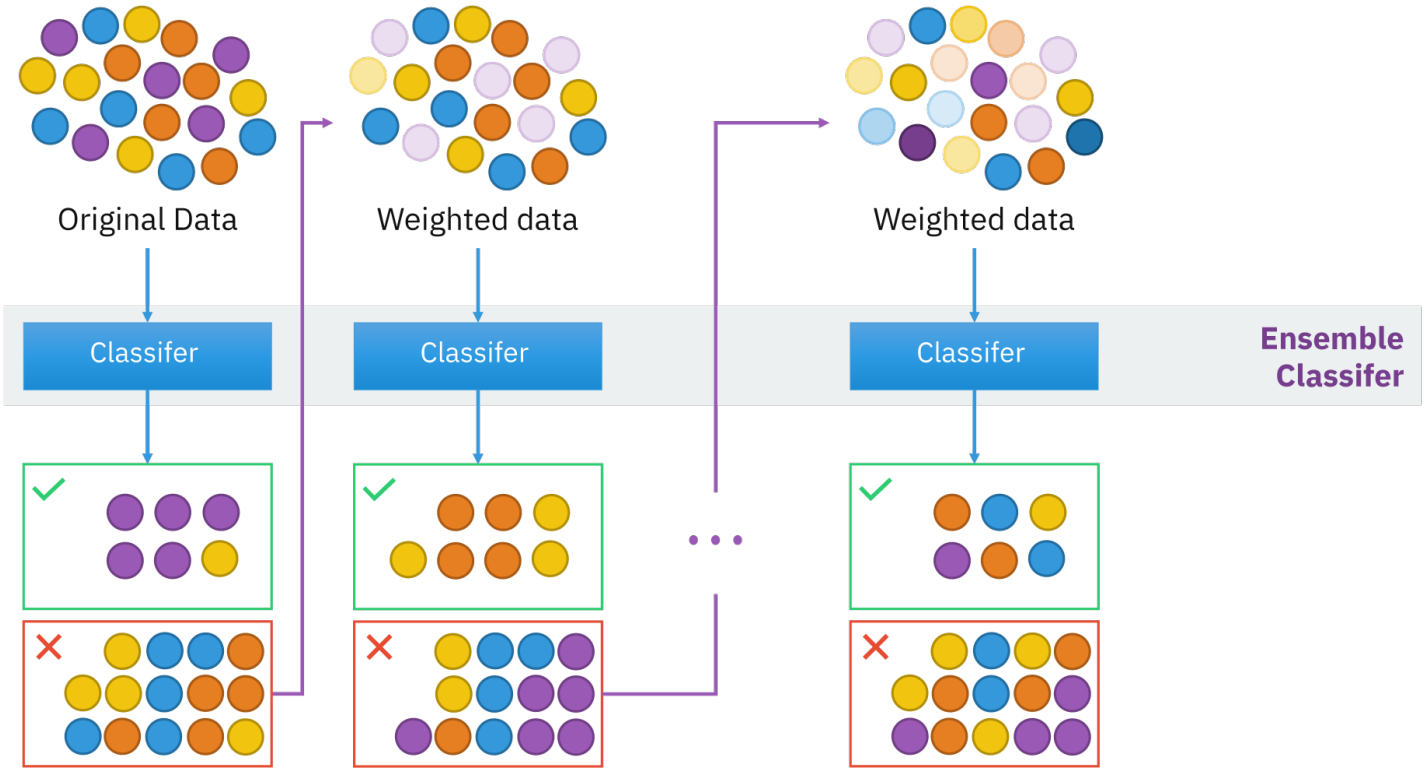


Figure Source. "An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset": https://en.wikipedia.org/wiki/Boosting_(machine_learning)

We know about the bagging method from the previous week which is the foundation for random forests, below we compare bagging with boosting.

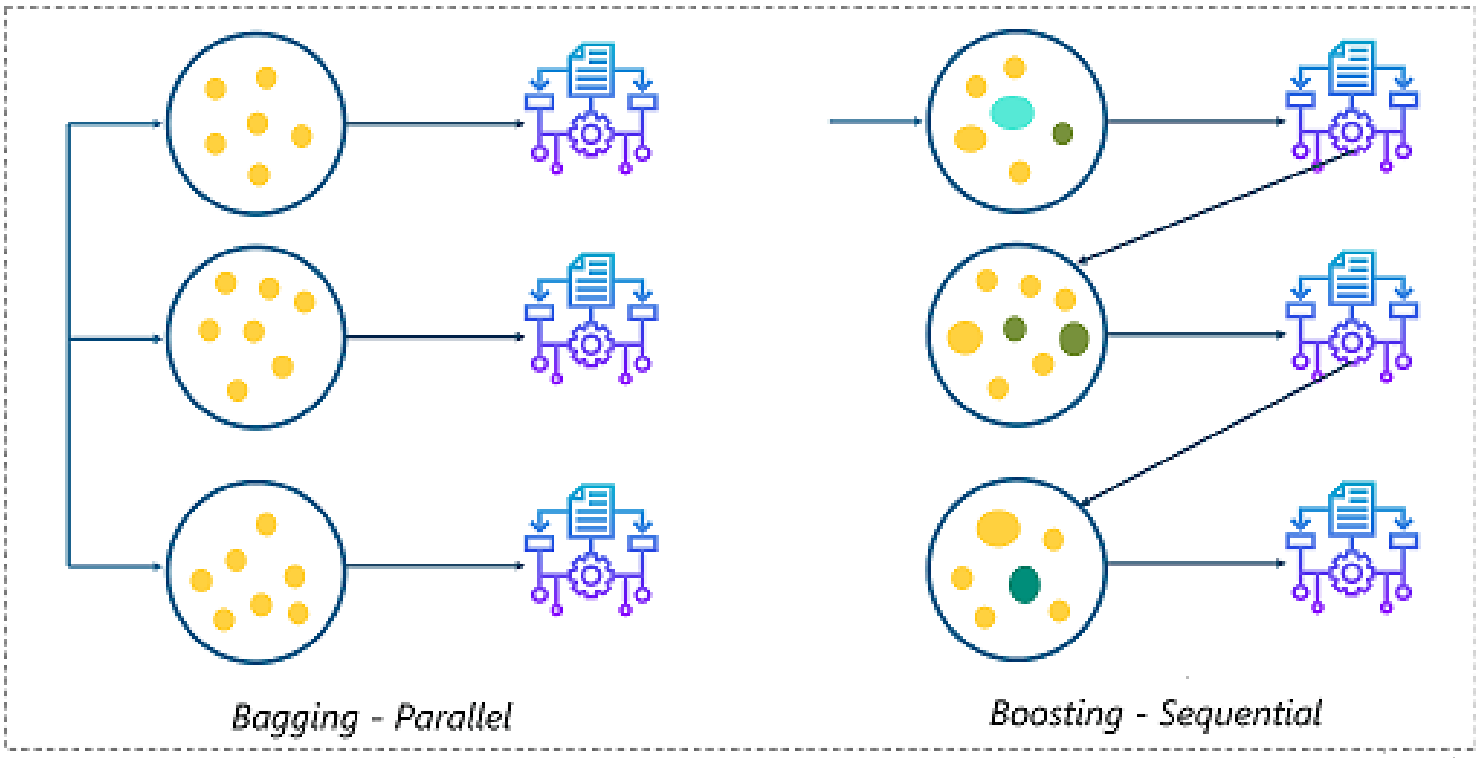


Figure Source. A Comprehensive Guide To Boosting Machine Learning Algorithms: https://www.edureka.co/blog/boosting-machine-learning/

Adaboost

AdaBoost focuses on misclassified training cases by the latest predictor and tries to guide a new predictor to pay more attention to the misclassified cases. The figure below explains how AdaBoost sequentially trains predictors for each iteration by updating weights of the misclassified training cases.

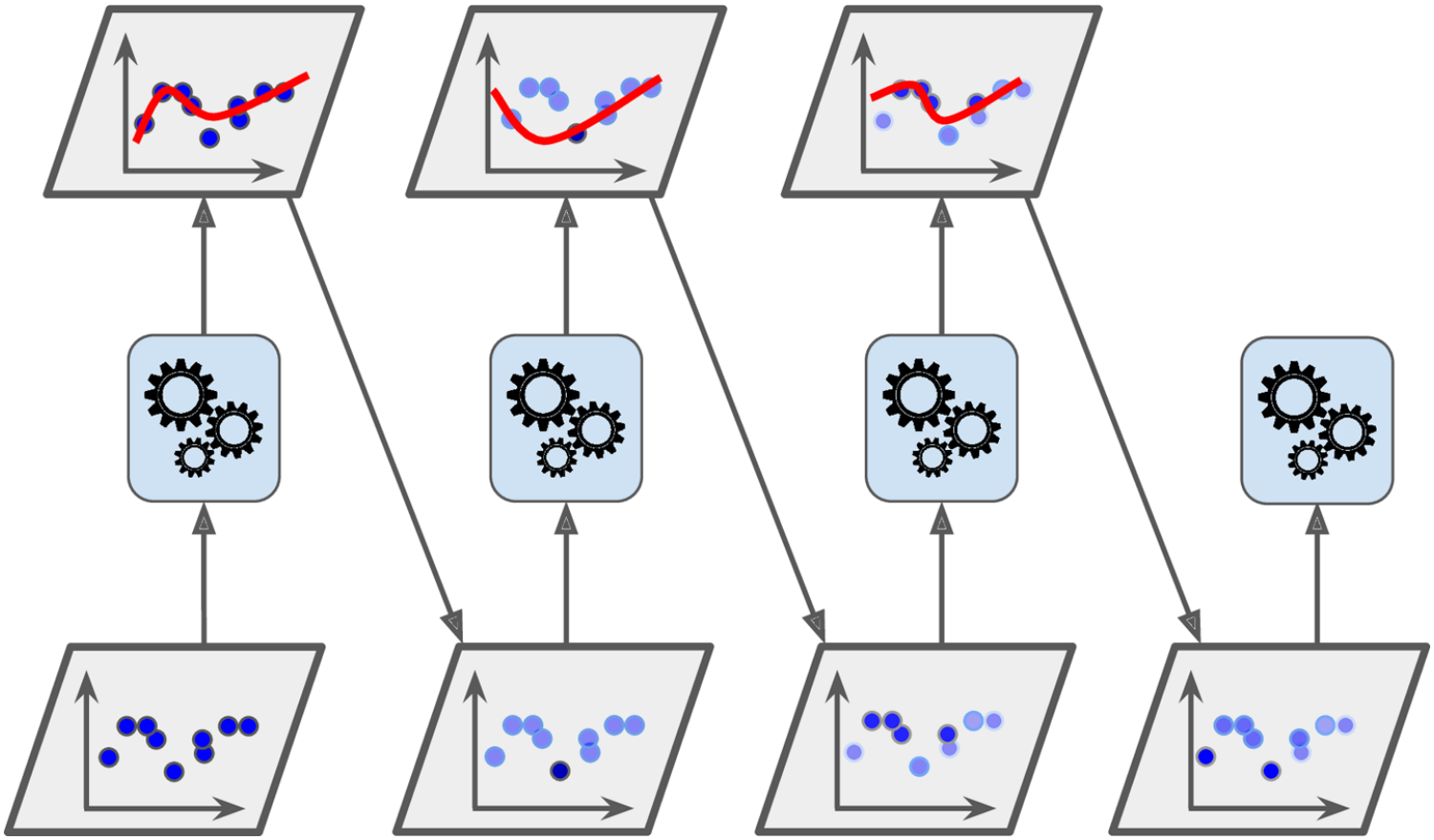


Figure: AdaBoost sequential training with instance weight updates. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Below the figure gives further details.

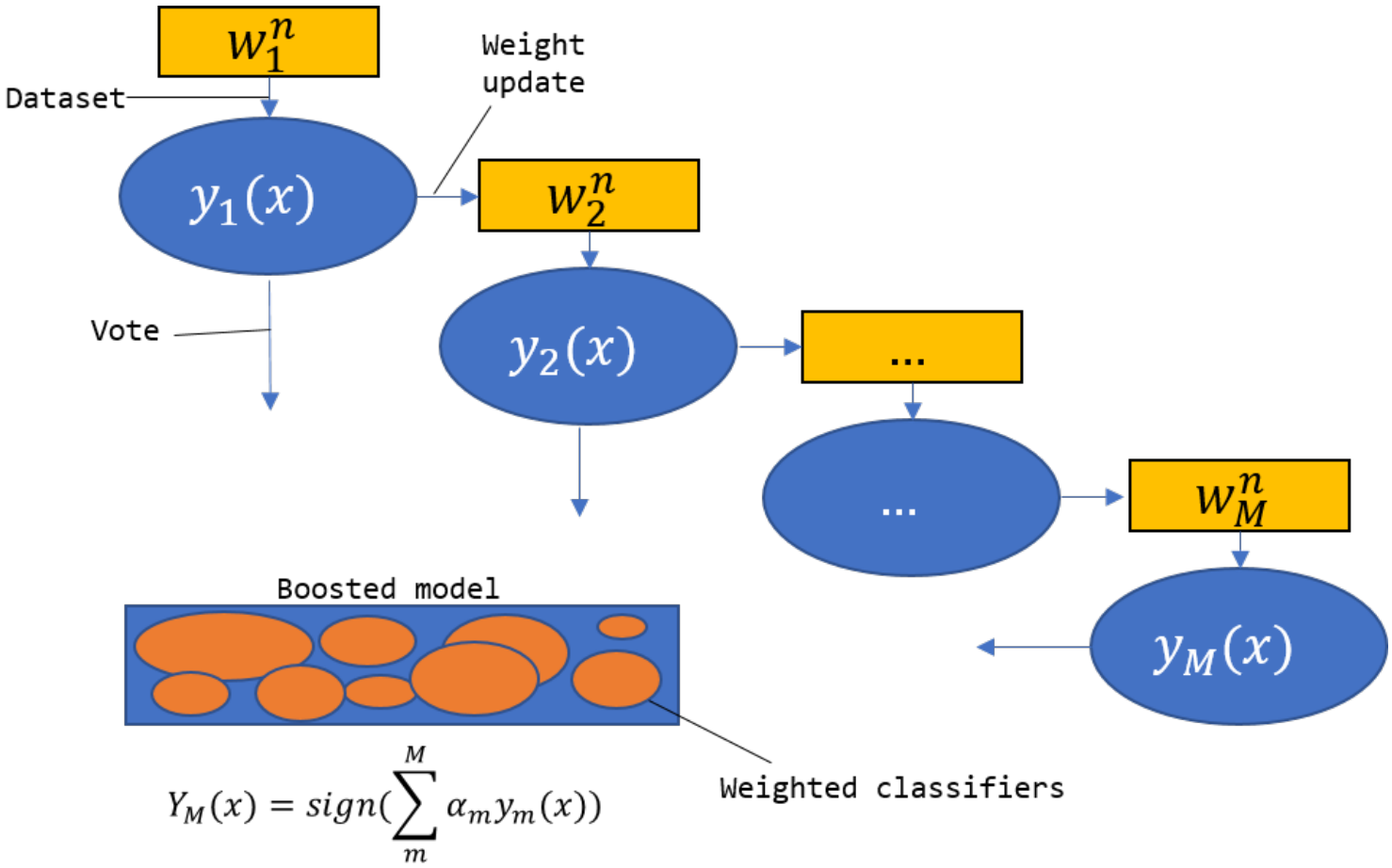


Figure Source. Boosting: https://www.python-course.eu/Boosting.php

The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data x, thereby producing a sequence of M weak classifiers as given below.

$$G_m(x), \text{ where } m = 1, 2, ..., M$$

Boosting employs stagewise additive modeling as shown below

$$F(x) = \sum_{m=1}^{M} \beta_m b(x; \lambda_m)$$

where $b(x; \lambda_m)$ is a tree and λ_m parameterizes the splits and the parameters (β_m, λ_m) are optimised in an iterative manner.

For simplicity, we transform the above by taking $f_m(x)$ as a function having m predictors and $f_{m-1}(x)$ as a function having $m - 1$ predictors as shown below

$$f_m(x) = f_{m-1}(x) + \alpha_m * G_m(x)$$

AdaBoost uses an exponential loss function given by

$$Err(f) = \sum_{i=1}^{N} e^{-y_i * f(x_i)}$$

AdaBoost uses {-1,1} as it's binary labels instead of {0,1}. We can expand the above error function by

$$Err(f) = \sum_{i=1}^{N} e^{-y_i f_{m-1}(x_i)} * e^{-y_i \alpha_m G_m(x_i)}$$

$$=> Err(\alpha_m, G_m) = \sum_{i=1}^{N} e^{-y_i f_{m-1}(x_i)} * e^{-y_i \alpha_m G_m(x_i)}$$

This can be rewritten as

$$Err = e^{-\alpha_m} \sum_{y_i = G_m(x_i)} w_i + e^{\alpha_m} \sum_{y_i \neq G_m(x_i)} w_i$$

or alternatively, we can rewrite as

$$Err = e^{-\alpha_m} (T_w - E_w) + e^{\alpha_m} E_w$$

where

$$T_w = \sum_{all y_i} w_i$$

$$E_w = \sum_{y_i \neq G_m(x_i)} w_i.$$

Then we differentiate the *Err* with respect to, α_m to get the minima point

$$\frac{dErr}{d\alpha_m} = -\alpha_m e^{-\alpha_m} (T_w - E_w) + \alpha_m e^{\alpha_m} E_w$$

Furthermore, equating d*Err*/dα_m = 0 we have

$$
\begin{aligned}
-\alpha_m e^{-\alpha_m}(T_w - E_w) + \alpha_m e^{\alpha_m} E_w &= 0 \\
=> e^{\alpha_m} E_w - e^{-\alpha_m}(T_w - E_w) &= 0 \\
=> e^{\alpha_m} E_w &= e^{-\alpha_m}(T_w - E_w) \\
=> e^{2\alpha_m} &= (Tw - Ew)/Ew \\
=> e^{2\alpha_m} &= \frac{1 - (E_w/T_w)}{(E_w/T_w)}
\end{aligned}
$$

We take log on both sides and get

$$\alpha_m = \frac{1}{2} log \frac{1 - err_m}{err_m}$$

where

$$err_m = \frac{E_w}{T_w} = \frac{\sum_{y_i \neq G_m(x_i)} w_i}{\sum w_i}$$

Further details: Rojas, R. (2009). AdaBoost and the super bowl of classifiers a tutorial introduction to adaptive boosting. *Freie University, Berlin, Tech. Rep:* http://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf

The use of the exponential loss in the context of additive modelling leads to the simple modular reweighting

The special form of the exponential loss function in AdaBoost leads to fitting trees to weighted versions of the original data, Instead of fitting trees to residuals.

Some important points to note:

1. We note that $G_m(x)$ only outputs $[-1, 1]$, *which* is scaled to some positive or negative value by multiplying with α_m. Hence, α_m is called confidence since we are showing that much faith on that specific predictor's output.
2. $\alpha_m G_m(x)$ will produce a series of positive/negative values and upon adding all of them, we have a sum which is either positive or negative. If it is positive then we say the output is 1 else it is -1.
3. $\alpha_m G_m(x)$ can be either positive or negative, and if more classifiers say that it is positive then the sum will turn out to be positive. We follow what the majority of classifiers vote, i.e. weighted voting.

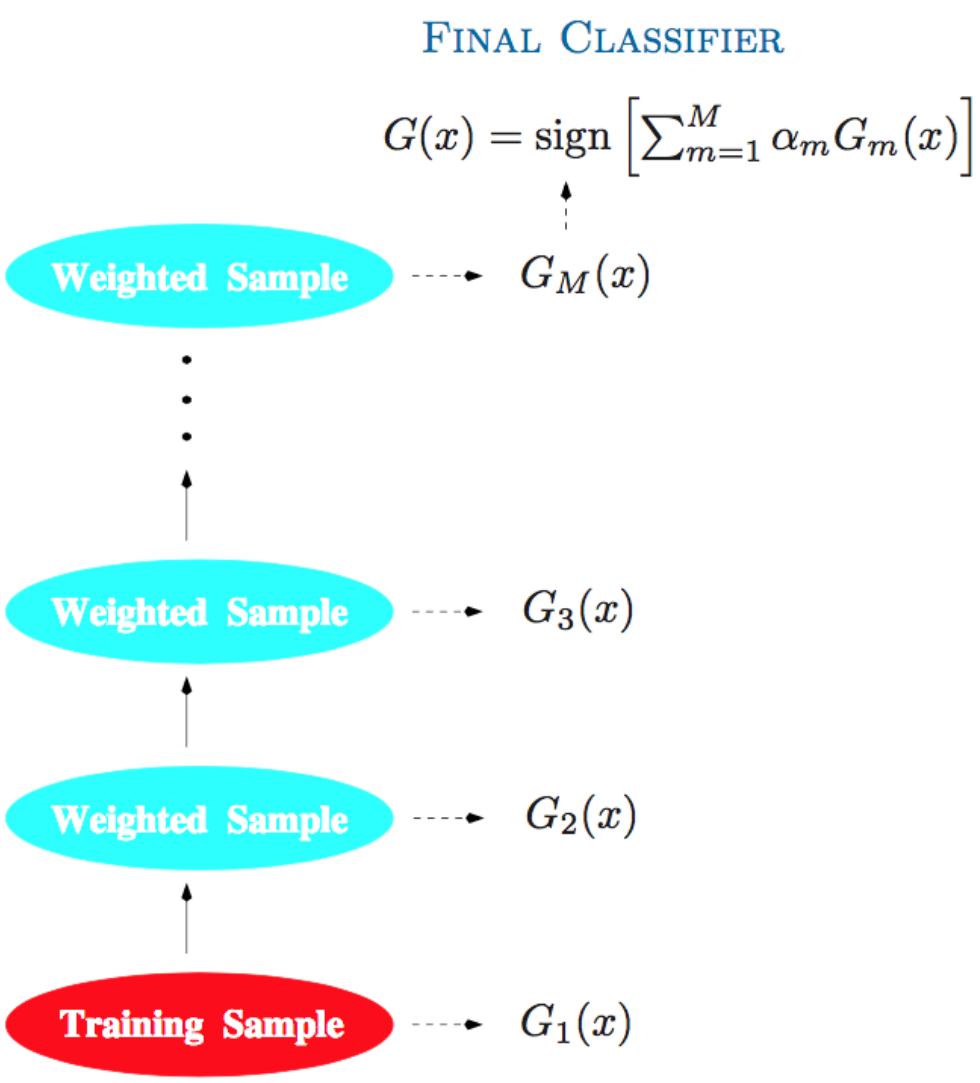


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Source: Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media. https://koalaverse.github.io/machine-learning-in-R/gradient-boosting-machines.html

Next we look at the Adaboost algorithm as shown below. The Adaboost algorithm features a training set containing m samples where all x inputs are an element of the total set X and y outputs are an element of a set comprising of only two values, i.e negative (-1) or positive class (+1).

Note the weights do not refer to the parameters in the model (such as neural networks) but as a means to select the subset of training data.

The algorithm begins by trains a base classifier and identifies training cases that are misclassified by

the first predictor. The algorithm now increases the relative weights of the misclassified training cases and trains a second predictor. Again the weights of the misclassified training examples by the second predictors are adjusted to train a third predictor, and so on. Note the weights do not refer to the parameters in the model (such as neural networks) but as a means to select the subset of training data.

Depending on the accuracy of the weighted training set, each predictor is assigned a weight. The overall strategy of making predictions is similar to bagging or pasting, except that different weights of the predictors are considered.

Algorithm 10.1 *AdaBoost.M1.*

---

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Source: Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media. https://koalaverse.github.io/machine-learning-in-R/gradient-boosting-machines.html

Considering the framework is essentially sequential, it's not easy to parallelise execution and use multiple CPU cores. Therefore, AdaBoost does not scale like bagging which can be easily parallelized.

Example

A tree with only two leaves is known as a stump. Adaboost uses stumps to generate strong leaners from weak learnings by updated the subset of the data over time. The figure below shows and example of a stump.

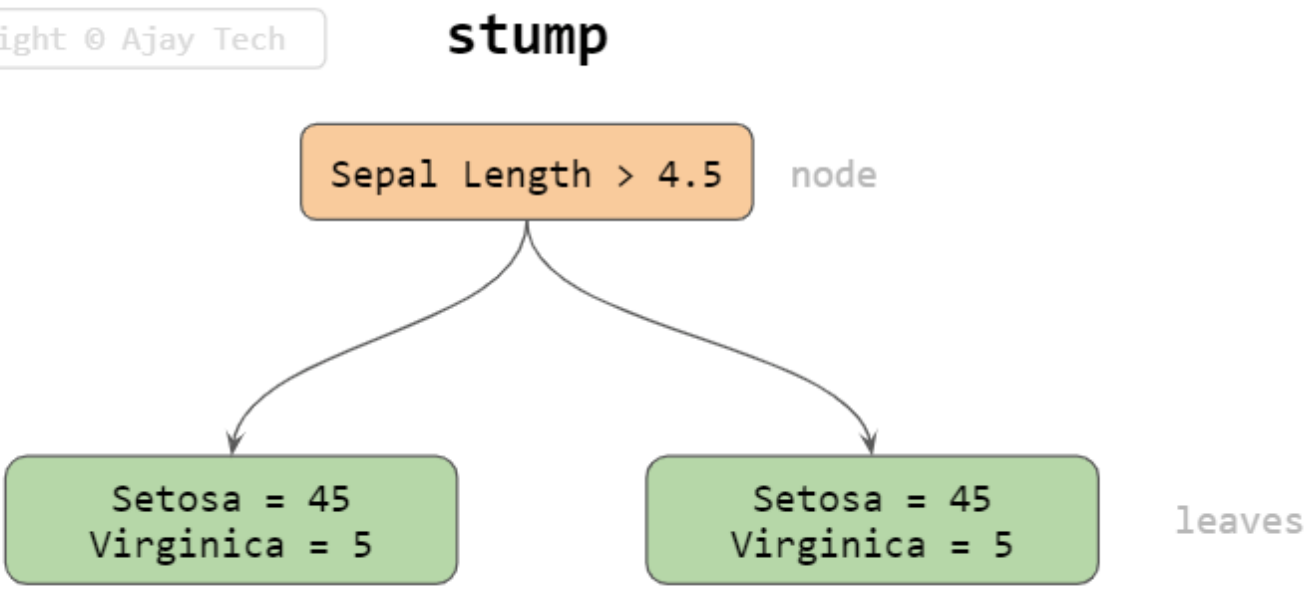


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Random Forest vs AdaBoost

We note that a Random Forest is a forest of trees which are fully grown while AdaBoost is a forest of stumps. In the diagram above, the stump just divides the dataset into two parts; where, sepal length > 4.5 and sepal length <= 4.5

It is reasonable to question the validity of a stump since it just divides the dataset into 2 parts. We can question how can a forest of stumps help build a complicated model that can compete with Random Forests?

The key principle behind Adaboost is that weak learners come together to form strong learners. principal

We can recall that in Random Forest ensemble, each tree is made independent of each other and the order of the trees doesn't matter. Once the RAndom Forest ensemble is created with bagging, the class that most trees in the forest vote is chosen a winner (in case of classification)

On the other hand, not all stumps in the AdaBoost algorithm have equal weightage.

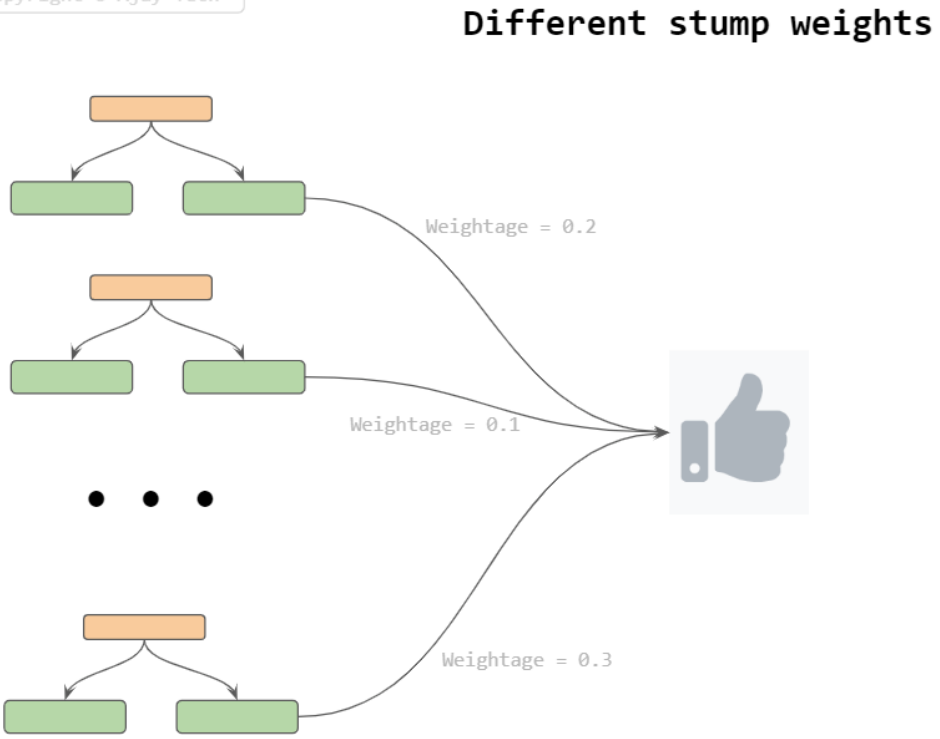


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

In the Iris classification problem, there are 4 possible stumps to start with since we have 4 attributes in the dataset. As done in Decision Trees, each stump would have a Gini index and the best one wins. In the following case, the stump that splits on petal length wins.

Sepal length	Sepal width	Petal length	Petal width	species	weight
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1

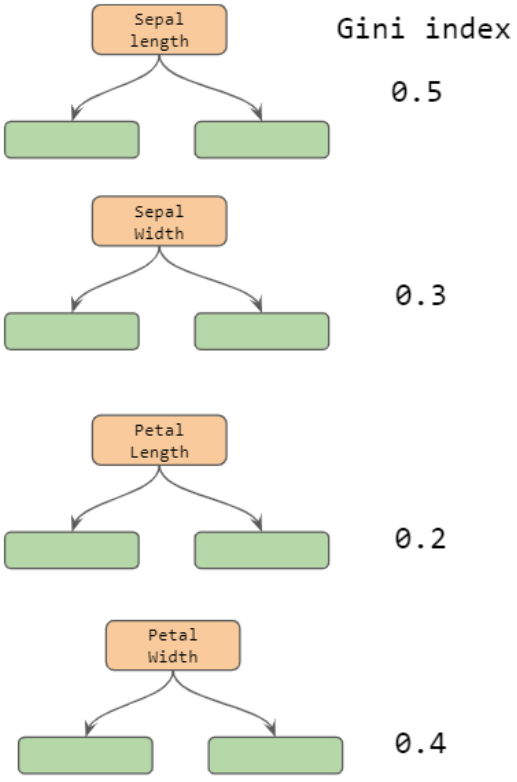


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Next, the winner stump is used to classify all the instances in the training dataset. Then for each instance, the algorithm checks if its correctly or incorrectly classified.

The general idea is to create another dataset based on the classification of the instances where those that are wrongly classified are given more chance to be included in the dataset.

The creation of new dataset is done based on a weightage scheme where those that are incorrectly classified are given an increased weightage. The weightage is based on a specific formula. Let's look at the figure below for an overview.

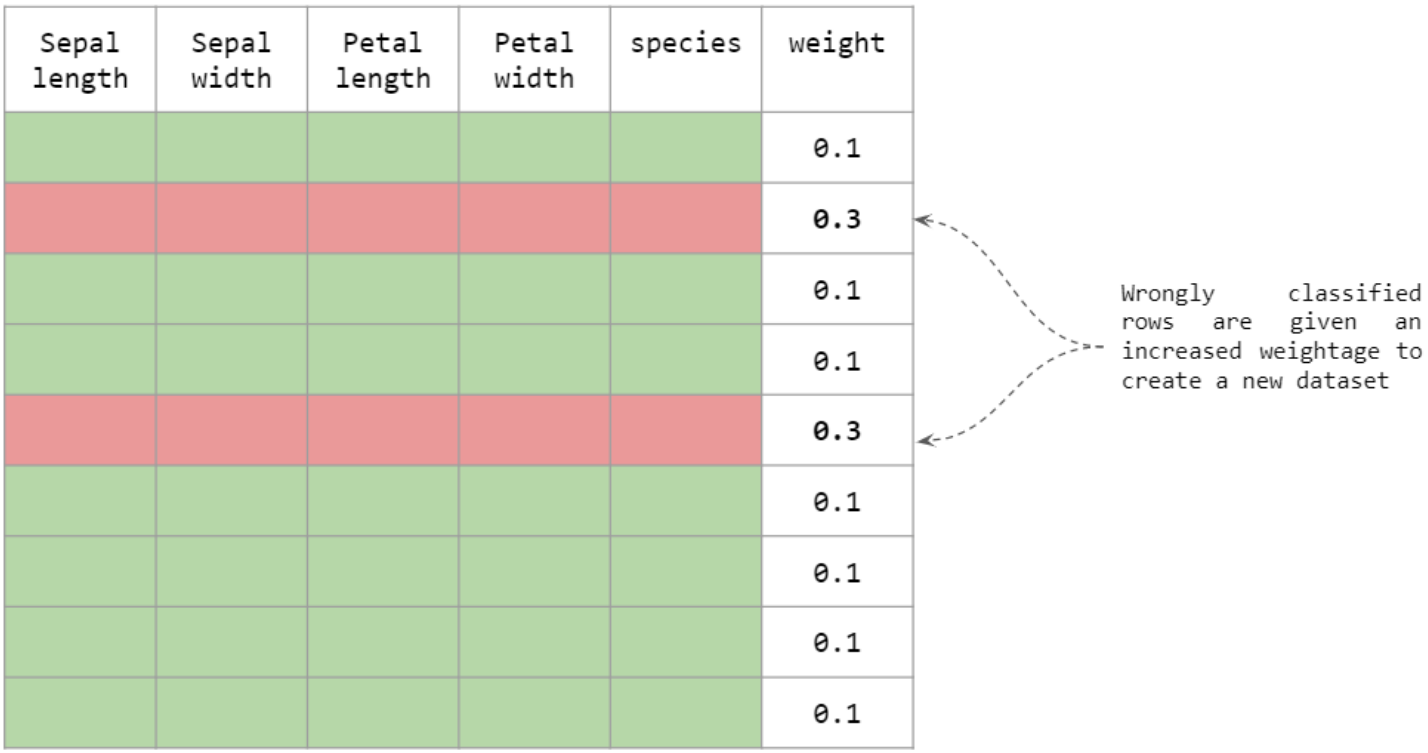


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

We then create a new dataset of the same size as the first, but select the data based on the weights. The instances with more weights will get more preference. Ultimately, the new dataset will have more rows thnt has misclassified labels based on the first stump's classification. See the figure below.

Sepal length	Sepal width	Petal length	Petal width	species	weight
					0.1
					0.3
					0.1
					0.1
					0.3
					0.1
					0.1
					0.1
					0.1

New dataset

Sepal length	Sepal width	Petal length	Petal width	species

Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Since the new dataset has more misclassified instances based on the first stump, the second stump (based on this new data set) will correctly be able to identify the wrongly classified ones, based on Gini index. Then the second stump is used for classification of the instances in the second dataset.

The weights in the new dataset are reset and then new weights will be computed based on the classification of the second stump.

Sepal length	Sepal width	Petal length	Petal width	species	weight
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1

Set to equal weightage again

Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Below the figure shows how the Gini index is calculated and the second stump is created. This time, its the Sepal width that has best Gini index.

Sepal length	Sepal width	Petal length	Petal width	species	weight
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1
					0.1

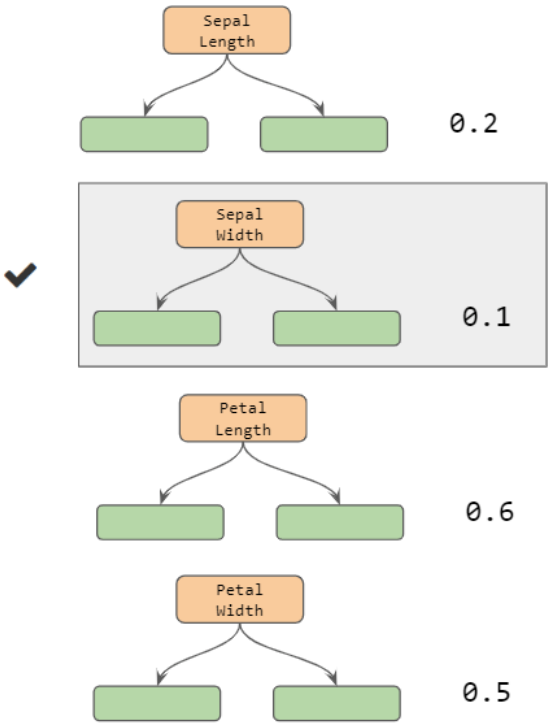


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

This procedure continues until a set of winning stumps get created each with particular or user-defined weightage. The sum of the weighted vote determines the final classification as shown below.

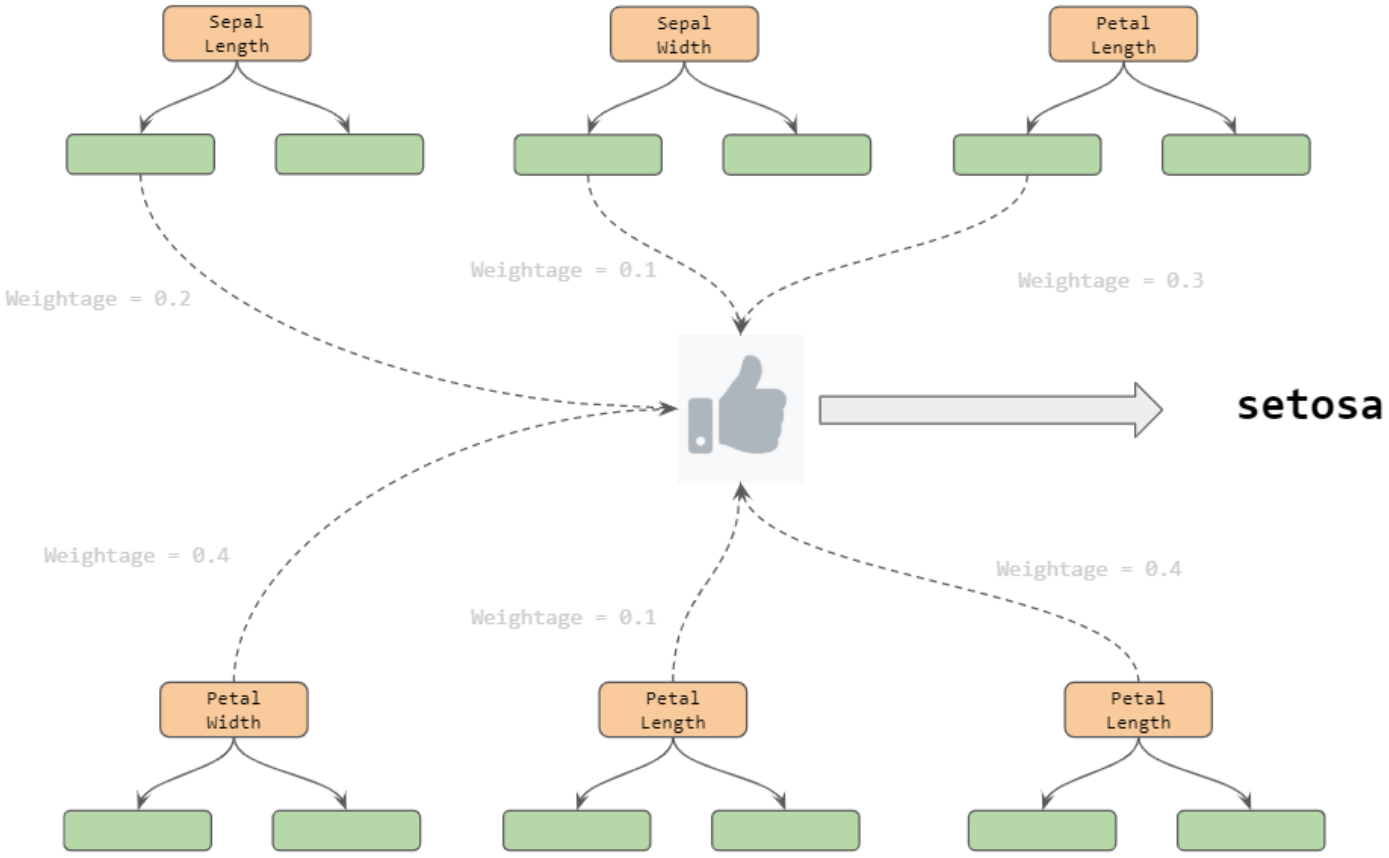


Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Below is an animation that shows the prediction for a regression problem. Note that in the beginning, weak classifiers are created and then the procedure repeats and there is lower error towards the end.

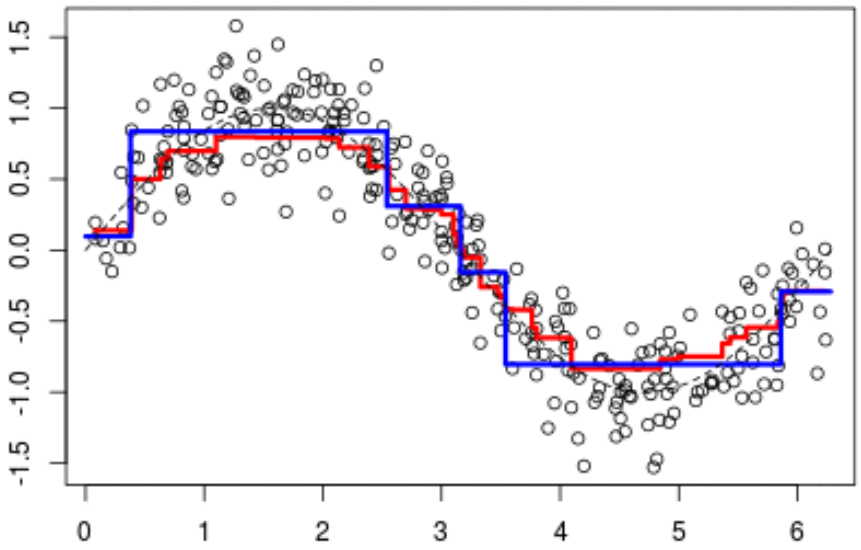
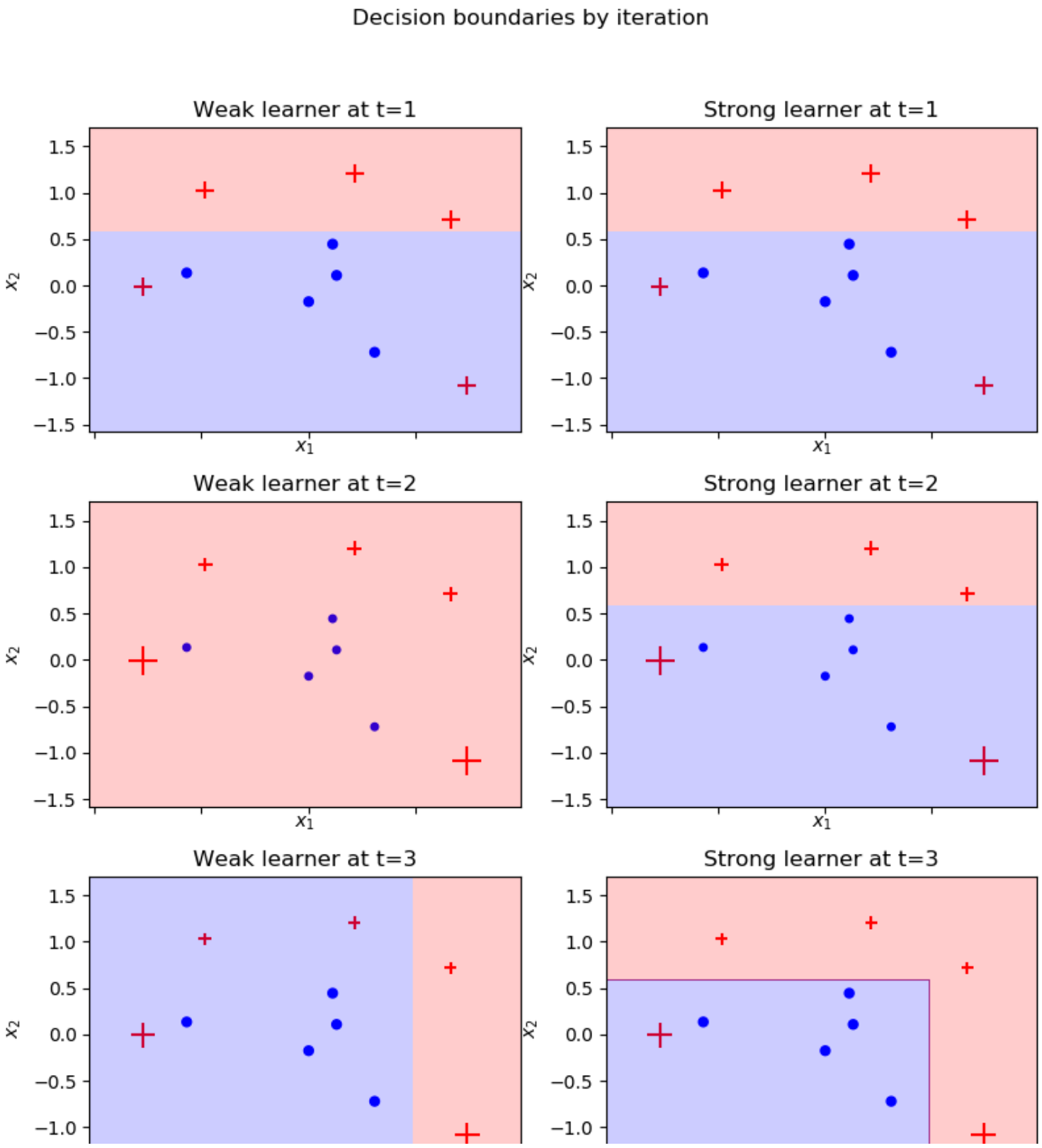
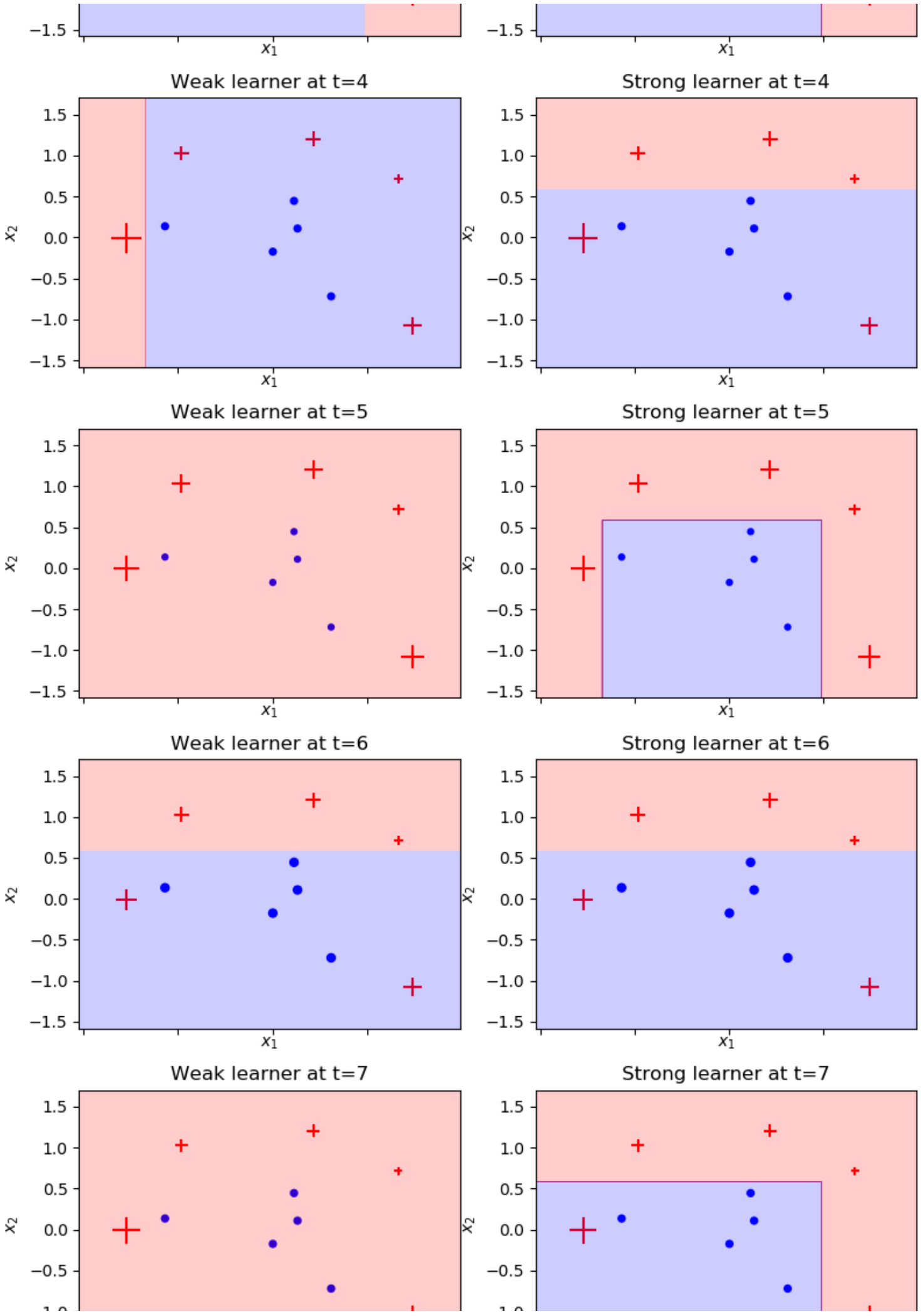


Figure Source: https://www.r-bloggers.com/2018/06/classification-from-scratch-boosting-11-8/

Below is a visualisation of how the decision boundary changes over time by the weak and strong learner at every iteration of the boosting process.




Weak learner at t=4
Strong learner at t=4
Weak learner at t=5
Strong learner at t=5
Weak learner at t=6
Strong learner at t=6
Weak learner at t=7
Strong learner at t=7
x_1
x_2

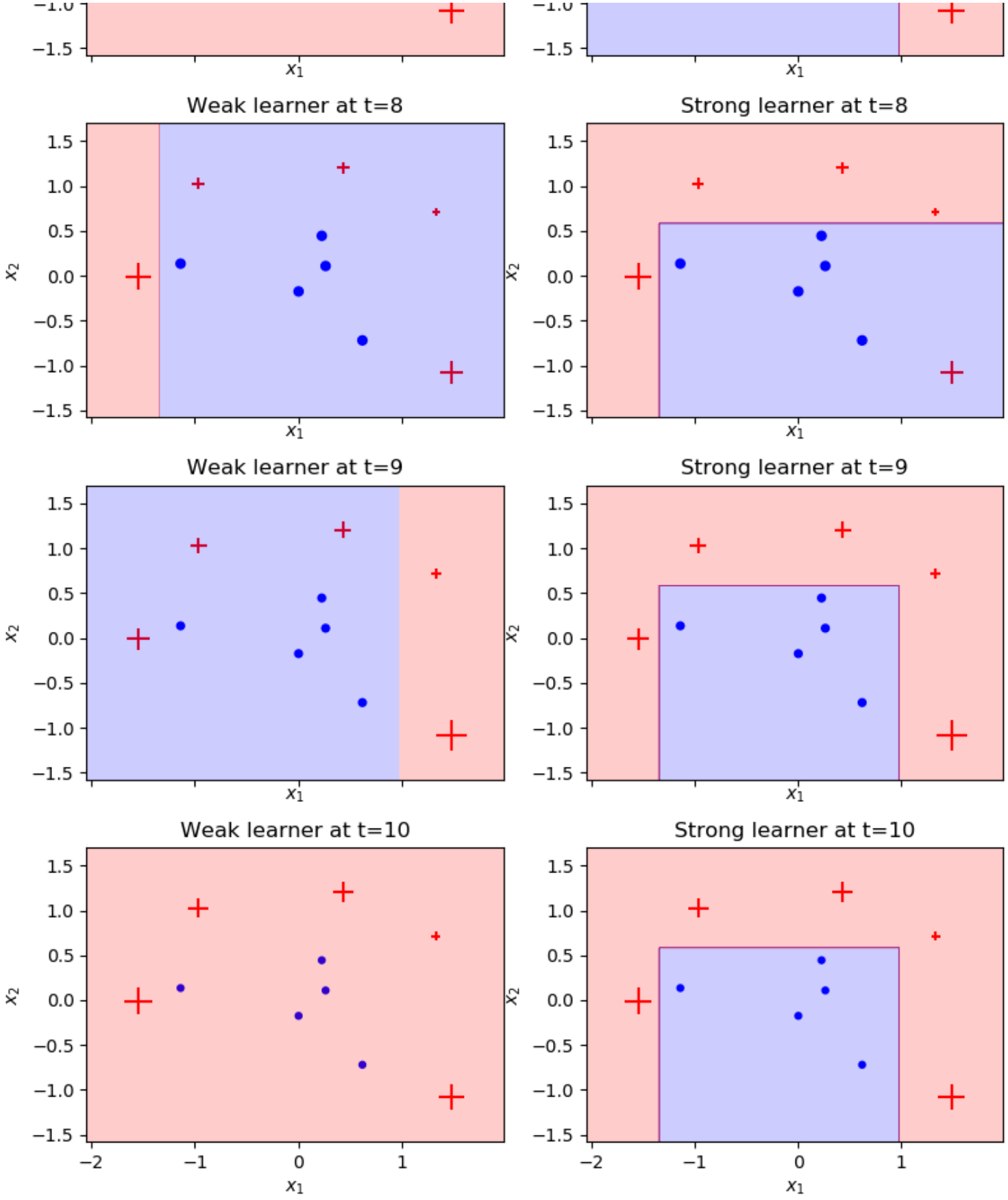


Figure Source. Building an AdaBoost classifier from scratch in Python: https://geoffruddock.com/adaboost-from-scratch-in-python/

In the above figure, note that our weak learners at iterations t=2,5,7,10 classify all points as positive.

This is due to the dataset at those particular stages where the lowest error is achieved by simply predicting all data points to be positive. In these cases, the negative samples are surrounded by proportionally higher-weighted positive samples.

In these cases, since we are only using stumps, there is no way to draw a linear decision boundary to correctly classify any number of negative data points without misclassifying a higher cumulative weight of positive samples.

All the negative instances are misclassified and therefore increasing in sample weight so that they are chosen in next dataset. Hence, the next iteration's weak learner discovers a meaningful decision boundary.

Another example

Data set

We are going to work on the following data set where each instance is represented as 2-dimensional space and we also have its class value.

Initially, we distribute weights in uniform distribution by **1/N** where N is the total number of instances.

Note that **weight*** stores **weight** times **actual** value for each instance. We use **weight*** as target value whereas **x1** and **x2** are features to build a decision stump. See figure below that describes the data and the **weight, weight*, prediction, loss**, and **weight* times loss**.

x1	x2	actual	weight	weight*
2	3	1	0.1	0.1
2	2	1	0.1	0.1
4	6	1	0.1	0.1
4	3	-1	0.1	-0.1
4	1	-1	0.1	-0.1
5	7	1	0.1	0.1
5	3	-1	0.1	-0.1
6	5	1	0.1	0.1
8	6	-1	0.1	-0.1
8	2	-1	0.1	-0.1

x1	x2	actual	weight	weight*	prediction	loss	weight * loss
2	3	1	0.1	0.1	1	0	0
2	2	1	0.1	0.1	1	0	0
4	6	1	0.1	0.1	-1	1	0.1
4	3	-1	0.1	-0.1	-1	0	0
4	1	-1	0.1	-0.1	-1	0	0
5	7	1	0.1	0.1	-1	1	0.1
5	3	-1	0.1	-0.1	-1	0	0
6	5	1	0.1	0.1	-1	1	0.1
8	6	-1	0.1	-0.1	-1	0	0
8	2	-1	0.1	-0.1	-1	0	0

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

The sum of **weight* loss** column stores the total error or **epsilon** which is 0.3 in the above table.

Next we define a new variable **alpha** as given below

$$alpha = ln[(1 - epsilon)/epsilon]/2$$

$$= \ln[(1 - 0.3)/0.3] / 2$$

$$= 0.42$$

We use alpha to update weights in the next round with

$$w_{i+1} = w_i * exp(-alpha * actual * prediction)$$

where i refers to instance number and sum of weights must be equal to 1. Hence, we normalize weight values (**norm(w_(i+1))**) and then **Round 1** is over

x1	x2	actual	weight	prediction	w_(i+1)	norm(w_(i+1))
2	3	1	0.1	1	0.065	0.071
2	2	1	0.1	1	0.065	0.071
4	6	1	0.1	-1	0.153	0.167
4	3	-1	0.1	-1	0.065	0.071
4	1	-1	0.1	-1	0.065	0.071
5	7	1	0.1	-1	0.153	0.167
5	3	-1	0.1	-1	0.065	0.071
6	5	1	0.1	-1	0.153	0.167
8	6	-1	0.1	-1	0.065	0.071
8	2	-1	0.1	-1	0.065	0.071

x1	x2	actual	weight	weight*
2	3	1	0.071	0.071
2	2	1	0.071	0.071
4	6	1	0.167	0.167
4	3	-1	0.071	-0.071
4	1	-1	0.071	-0.071
5	7	1	0.167	0.167
5	3	-1	0.071	-0.071
6	5	1	0.167	0.167
8	6	-1	0.071	-0.071
8	2	-1	0.071	-0.071

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Round 2:

We shift **norm(w_(i+1))** column to weight column in this round and build a decision stump. We again computer the prediction, loss, weight *loss as shown below.

Then we compute the following:

epsilon = 0.21, alpha = 0.65

x1	x2	actual	weight	prediction	loss	weight * loss
2	3	1	0.071	-1	1	0.071
2	2	1	0.071	-1	1	0.071
4	6	1	0.167	1	0	0
4	3	-1	0.071	-1	0	0
4	1	-1	0.071	-1	0	0
5	7	1	0.167	1	0	0
5	3	-1	0.071	-1	0	0
6	5	1	0.167	1	0	0
8	6	-1	0.071	1	1	0.071
8	2	-1	0.071	-1	0	0

x1	x2	actual	weight	prediction	w_(i+1)	norm(w_(i+1))
2	3	1	0.071	-1	0.137	0.167
2	2	1	0.071	-1	0.137	0.167
4	6	1	0.167	1	0.087	0.106
4	3	-1	0.071	-1	0.037	0.045
4	1	-1	0.071	-1	0.037	0.045
5	7	1	0.167	1	0.087	0.106
5	3	-1	0.071	-1	0.037	0.045
6	5	1	0.167	1	0.087	0.106
8	6	-1	0.071	1	0.137	0.167
8	2	-1	0.071	-1	0.037	0.045

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Round 3

We skip the details and only show the computations of interest. Hence:

epsilon = 0.31, alpha = 0.38

x1	x2	actual	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.167	1	0	0	0.114	0.122
2	2	1	0.167	1	0	0	0.114	0.122
4	6	1	0.106	-1	1	0.106	0.155	0.167
4	3	-1	0.045	-1	0	0	0.031	0.033
4	1	-1	0.045	-1	0	0	0.031	0.033
5	7	1	0.106	-1	1	0.106	0.155	0.167
5	3	-1	0.045	-1	0	0	0.031	0.033
6	5	1	0.106	-1	1	0.106	0.155	0.167
8	6	-1	0.167	-1	0	0	0.114	0.122
8	2	-1	0.045	-1	0	0	0.031	0.033

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Round 4

We skip the details and only show the computations of interest.

epsilon = 0.10, alpha = 1.10

x1	x2	actual	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.122	1	0	0	0.041	0.068
2	2	1	0.122	1	0	0	0.041	0.068
4	6	1	0.167	1	0	0	0.056	0.093
4	3	-1	0.033	1	1	0.033	0.1	0.167
4	1	-1	0.033	1	1	0.033	0.1	0.167
5	7	1	0.167	1	0	0	0.056	0.093
5	3	-1	0.033	1	1	0.033	0.1	0.167
6	5	1	0.167	1	0	0	0.056	0.093
8	6	-1	0.122	-1	0	0	0.041	0.068
8	2	-1	0.033	-1	0	0	0.011	0.019

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Prediction

The cumulative sum of each round (R1, R2 ..) alpha times prediction gives the final prediction:

Alpha	R1	R2	R3	R4
	0.42	0.65	0.38	1.1
Predictions	R1	R2	R3	R4
	1	-1	1	1
	1	-1	1	1
	-1	1	-1	1
	-1	-1	-1	1
	-1	-1	-1	1
	-1	1	-1	1
	-1	-1	-1	1
	-1	1	-1	1
	-1	1	-1	-1
	-1	-1	-1	-1

Figure Adapted: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

We can visualise the 4 rounds below.

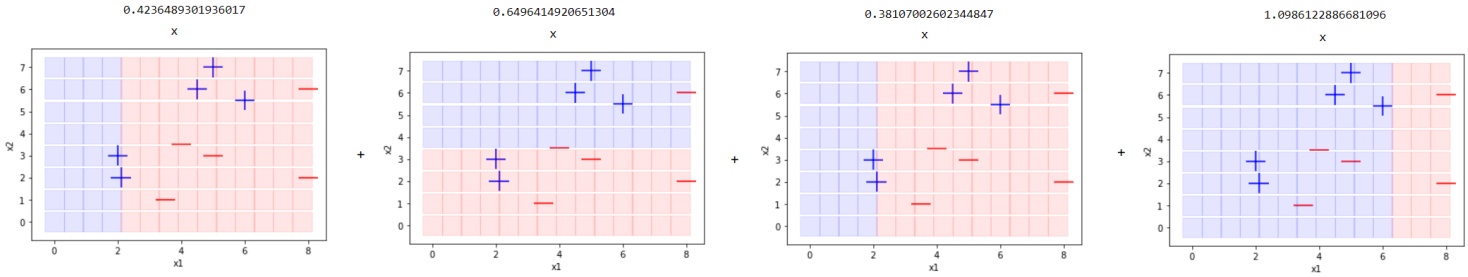


Figure Source: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

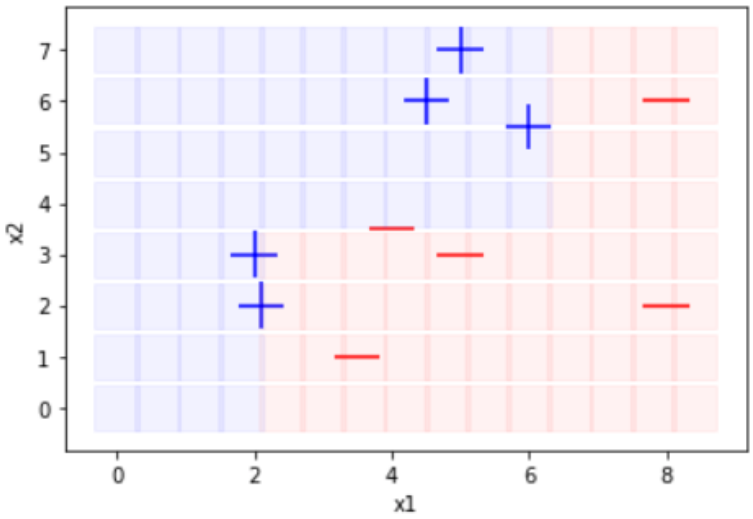


Figure Source: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Pruning

We notice that both round 1 and round 3 produce same results. Pruning in AdaBoost removes similar weak classifier to improve performance. Besides, we can increase the multiplier alpha value of remaining one. In this case, we remove round 3 and append its coefficient to round 1.

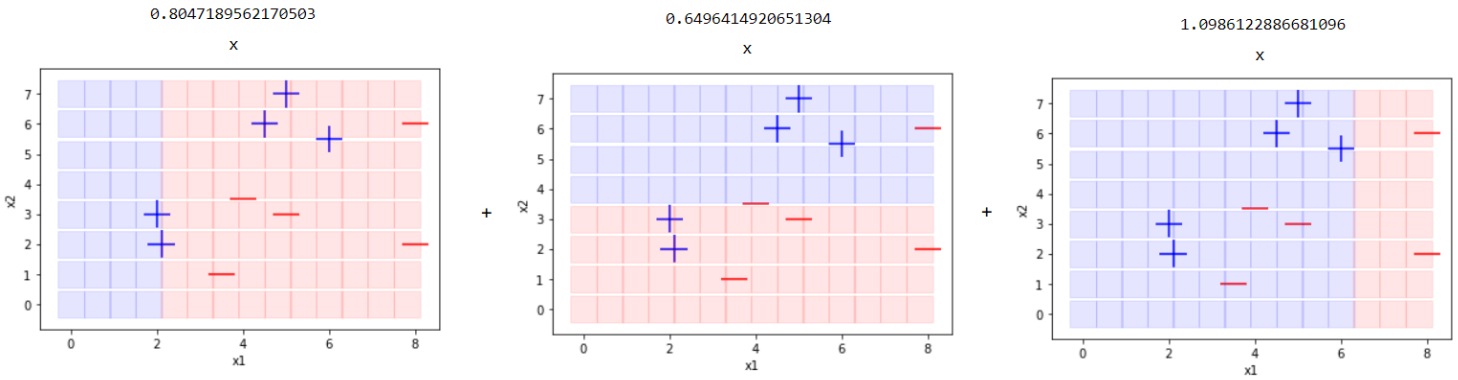


Figure Source: A Step by Step Adaboost Example: https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/

Next, we look at a Python implementation where the process is described in detail.

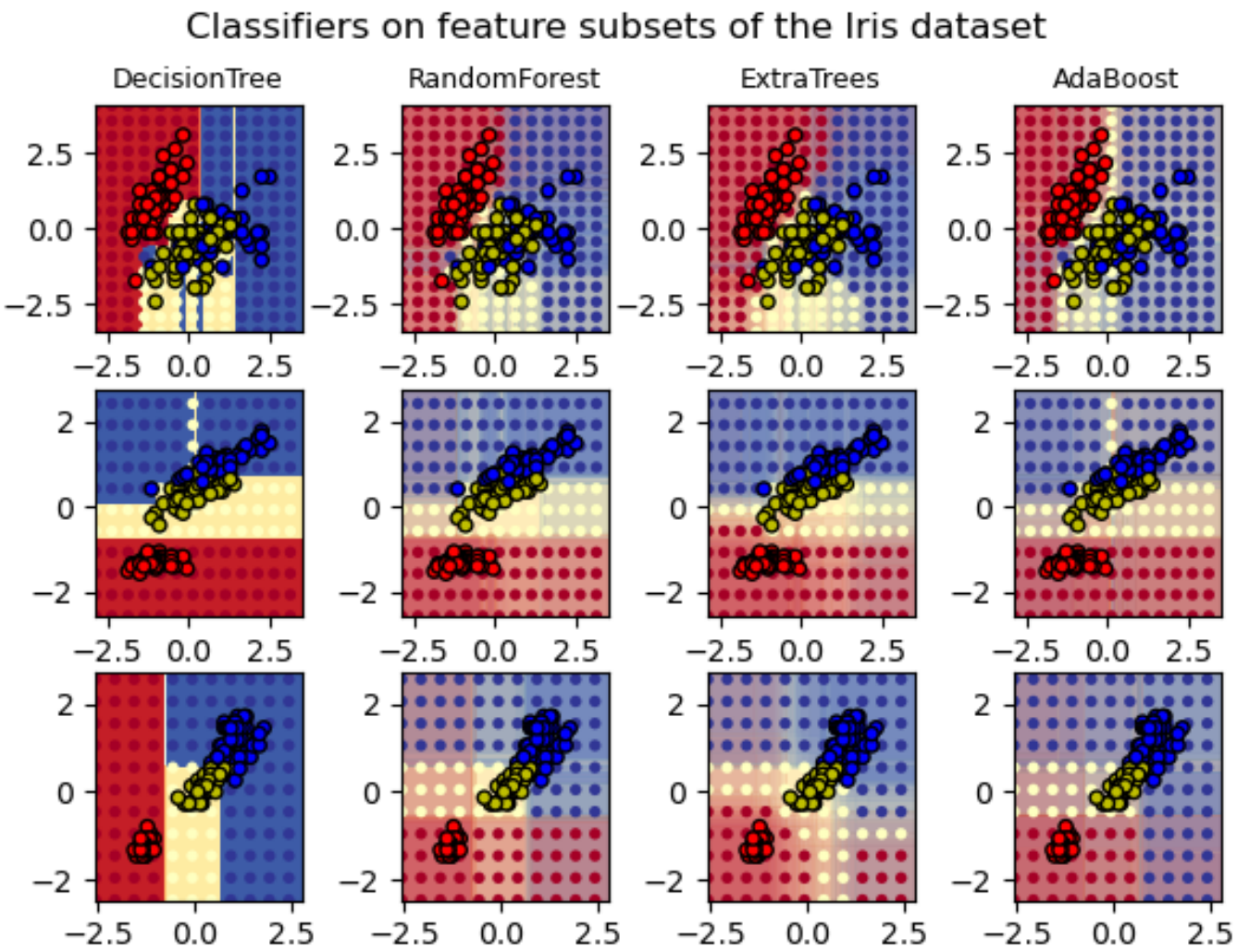
Code from Scratch

Run PYTHON

```python
import numpy as np

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import AdaBoostClassifier

def I(flag):
    return 1 if flag else 0

def sign(x):
    return abs(x)/x if x!=0 else 1

class AdaBoost:

```

Code Source: https://medium.com/analytics-vidhya/implementing-an-adaboost-classifier-from-scratch-e30ef86e9f1b

Finally, we compare the performance of Adaboost with Random Forest and Decision Trees.



TEXT

```
DecisionTree with features [0, 1] has a score of 0.9266666666666666
RandomForest with 30 estimators with features [0, 1] has a score of 0.92
ExtraTrees with 30 estimators with features [0, 1] has a score of 0.9266
AdaBoost with 30 estimators with features [0, 1] has a score of 0.853333
DecisionTree with features [0, 2] has a score of 0.9933333333333333
RandomForest with 30 estimators with features [0, 2] has a score of 0.99
ExtraTrees with 30 estimators with features [0, 2] has a score of 0.9933
AdaBoost with 30 estimators with features [0, 2] has a score of 0.993333
DecisionTree with features [2, 3] has a score of 0.9933333333333333
RandomForest with 30 estimators with features [2, 3] has a score of 0.99
ExtraTrees with 30 estimators with features [2, 3] has a score of 0.9933
AdaBoost with 30 estimators with features [2, 3] has a score of 0.993333
```

Figure Source. Plot the decision surfaces of ensembles of trees on the iris dataset: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_iris.html#sphx-glr-auto-examples-ensemble-plot-forest-iris-py

The above plot compares the decision surfaces learned by a decision tree classifier, a random forest classifier, an extra- trees classifier, and an AdaBoost classifier. In the first row, the classifiers are built using the sepal width and the sepal length features only, on the second row using the petal length and sepal length only, and on the third row using the petal width and the petal length only.

Scikit learn example

The following code segment uses `AdaBoostClassifier` class to train an AdaBoost classifier based on 100 Decision Trees. Scikit-learn uses a multiclass version of AdaBoost called *SAMME*, and its variant *SAMME.R* for 'real' values.

Run PYTHON

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Let's load data
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split dataset: 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

```

Figure Source. Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

Run R

```r
#https://www.datatechnotes.com/2018/03/classification-with-adaboost-mode


# Load libraries
library(mlbench)
library(caret)
library(caretEnsemble)

library(adabag)
library(caret)

indexes=createDataPartition(iris$Species, p=.90, list = F)
train = iris[indexes, ]
test = iris[-indexes, ]
```

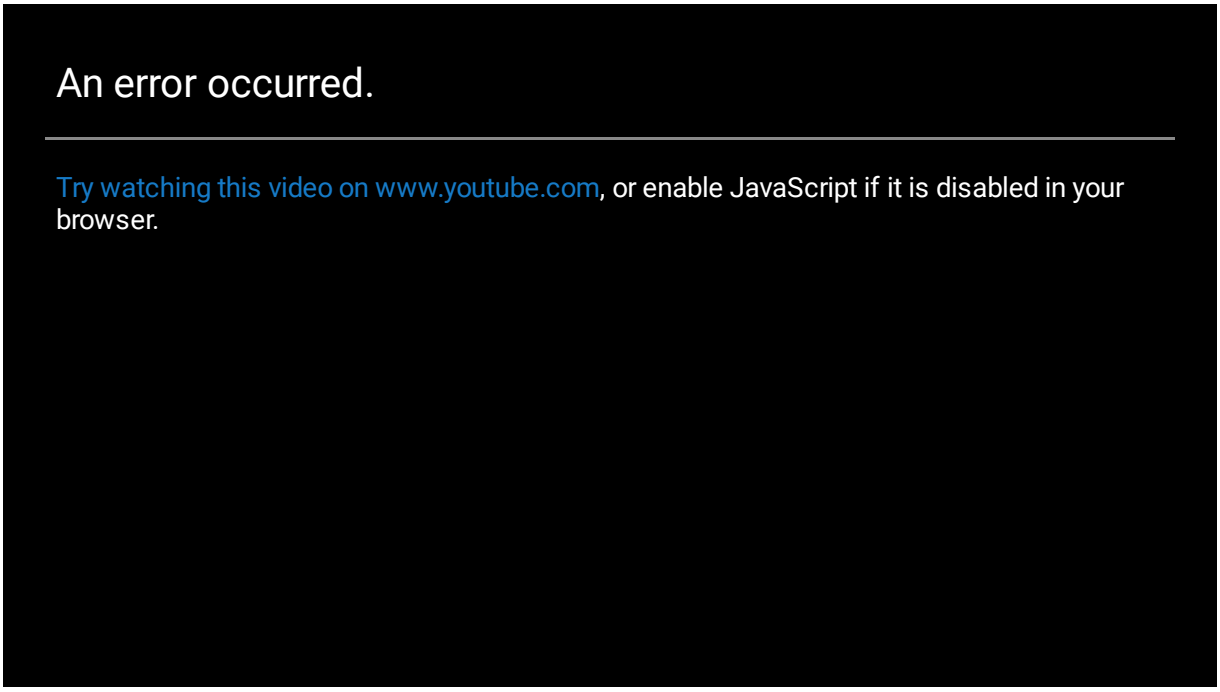
Code Source. https://www.datatechnotes.com/2018/03/classification-with-adaboost-model-in-r.html

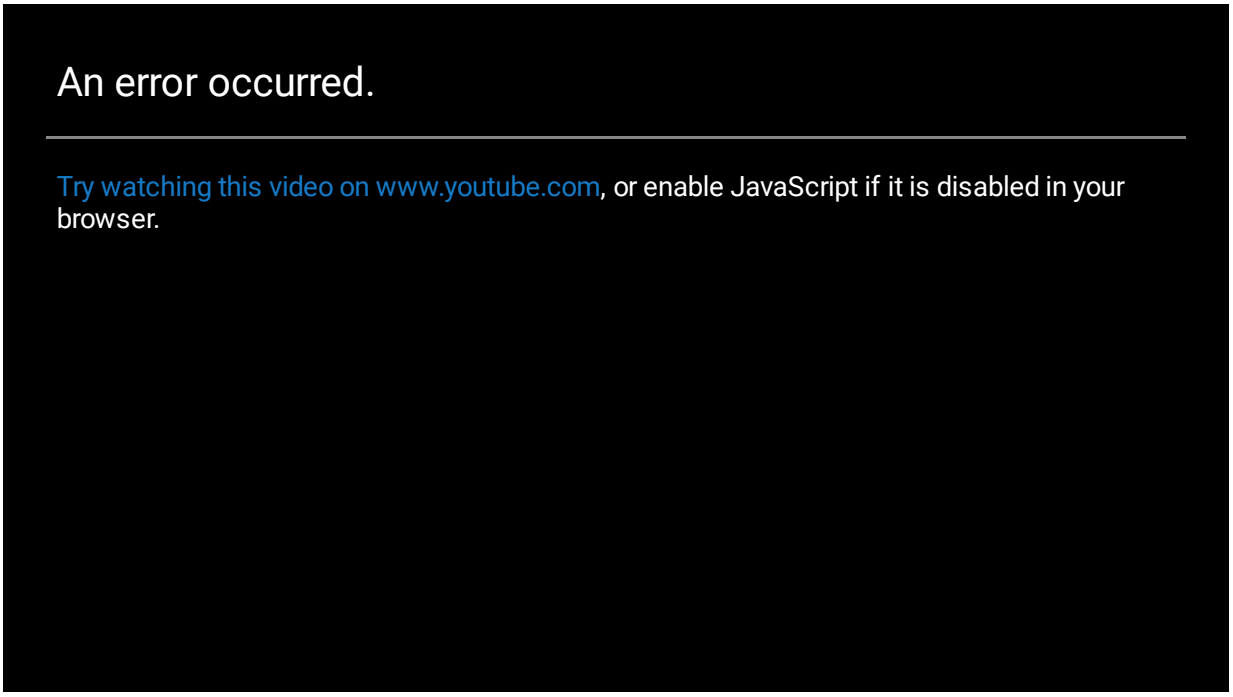
Run R

```r

# Load libraries
library(mlbench)
library(caret)
library(caretEnsemble)


# Load the dataset
data(Ionosphere)
dataset <- Ionosphere
dataset <- dataset[,-2]
dataset$V1 <- as.numeric(as.character(dataset$V1))

```

Adaboost in R https://machinelearningmastery.com/machine-learning-ensembles-with-r/

Some videos





Additional Resources

1. Lecture Notes from MIT course: http://people.csail.mit.edu/dsontag/courses/ml12/slides/lecture13.pdf
2. Lecture Notes from Toronto course:

https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf

References

1. Freund, Y., & Schapire, R. E. (1996, July). Experiments with a new boosting algorithm. In *icml* (Vol. 96, pp. 148-156): http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.6252&rep=rep1&type=pdf
2. Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, *4*(Nov), 933-969: http://www.jmlr.org/papers/volume4/freund03a/freund03a.pdf
3. Rätsch, G., Onoda, T., & Müller, K. R. (2001). Soft margins for AdaBoost. *Machine learning*, *42*(3), 287-320: https://link.springer.com/content/pdf/10.1023/A:1007618119488.pdf
4. Bartlett, P. L., & Traskin, M. (2007). Adaboost is consistent. *Journal of Machine Learning Research*, *8*(Oct), 2347-2368: http://www.jmlr.org/papers/volume8/bartlett07b/bartlett07b.pdf
5. Hastie, T., Rosset, S., Zhu, J., & Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, *2*(3), 349-360: https://www.intlpress.com/site/pub/files/_fulltext/journals/sii/2009/0002/0003/SII-2009-0002-0003-a008.pdf
6. Wang, L., Sugiyama, M., Yang, C., Zhou, Z. H., & Feng, J. (2008, July). On the margin explanation of boosting algorithms. In *COLT* (pp. 479-490): https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/colt08.pdf
7. Walach, E., & Wolf, L. (2016, October). Learning to count with cnn boosting. In *European conference on computer vision* (pp. 660-676). Springer: http://courses.cs.tau.ac.il/~wolf/papers/learning-count-cnn.pdf
8. Badirli, S., Liu, X., Xing, Z., Bhowmik, A., & Keerthi, S. S. (2020). Gradient Boosting Neural Networks: GrowNet. *arXiv preprint arXiv:2002.07971*. https://arxiv.org/pdf/2002.07971.pdf

Gradient Boosting

Gradient Boosting is an ensemble method that also uses an iterative framework similar to AdaBoost. However, instead of updating weights of misclassified cases and creating a new dataset for each iteration, Gradient Boosting calculates **residual errors** from the latest predictor and tries to fit the new predictor to these residual errors.

The residuals will decrease over time. Gradient Boosting provides a sequential tree where the base tree is boosted with time, through residual errors.

We provide a visualisation of the process as follows.

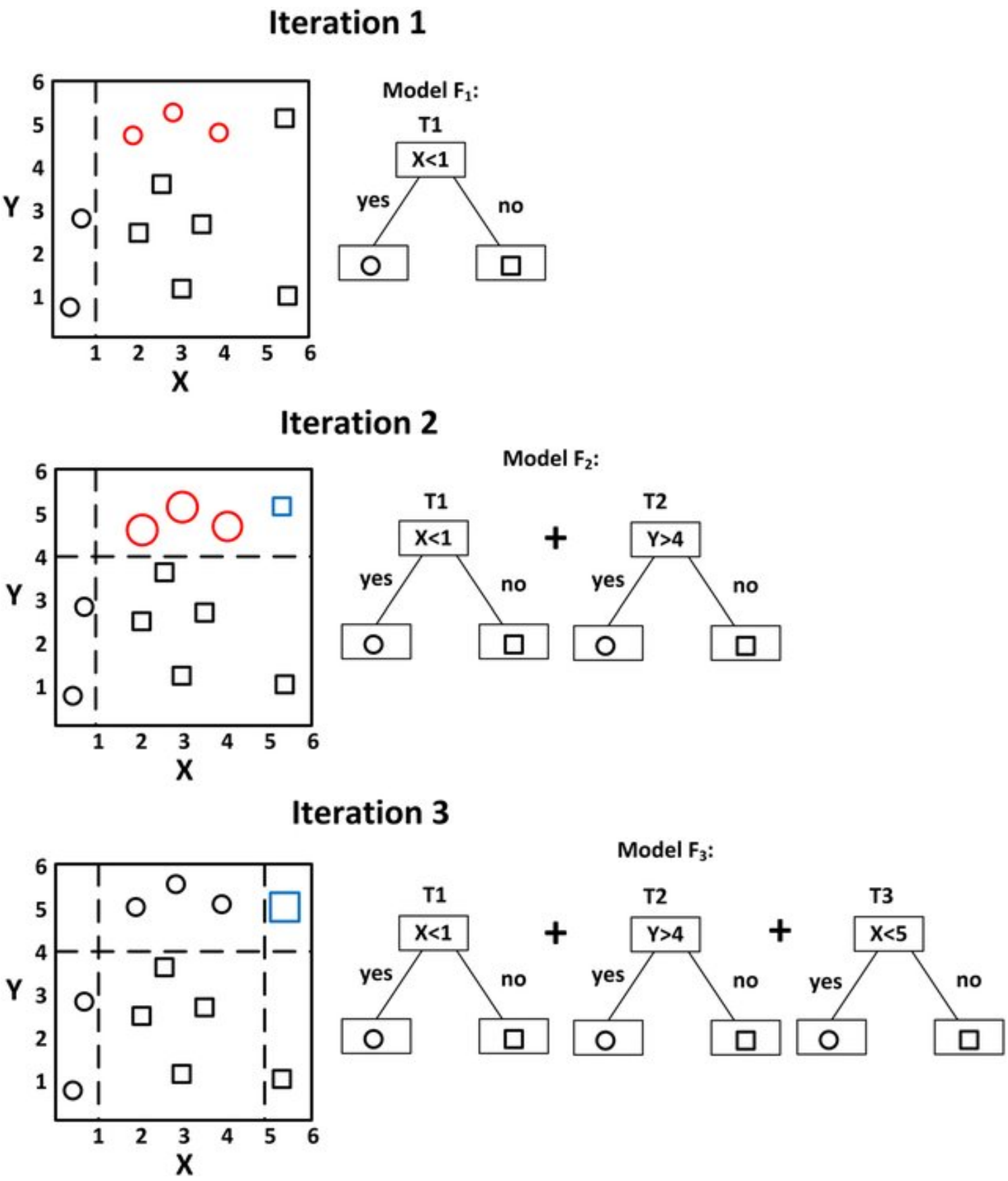


Figure Source.Zhang, Z., Mayer, G., Dauvilliers, Y., Plazzi, G., Pizza, F., Fronczek, R., ... & Da Silva, A. M. (2018). Exploring the clinical features of narcolepsy type 1 versus narcolepsy type 2 from European Narcolepsy Network database with machine learning. *Scientific reports*, *8*(1), 1-11.

In the above figure, note how the residuals from **iteration 1** are used for training stump in **iteration**

2 where new decision boundaries are created which eventually progresses to **iteration 3**.

In this case, unlike Adaboost, the dataset is not generated for every iteration.

Next, we look at Python code that archives this implementation. First, let's fit a `DecisionTreeRegressor` to the training set (for example, a noisy quadratic training set):

PYTHON

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

Next, we'll train a second `DecisionTreeRegressor` on the residual errors made by the first predictor:

PYTHON

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Then we train a third regressor on the residual errors made by the second predictor:

PYTHON

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Now we have an ensemble containing three trees. It can make predictions on a new instance simply by adding up the predictions of all the trees:

PYTHON

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree
```

The following figure represents the predictions of these three trees in the left column and the ensemble's predictions in the right column.

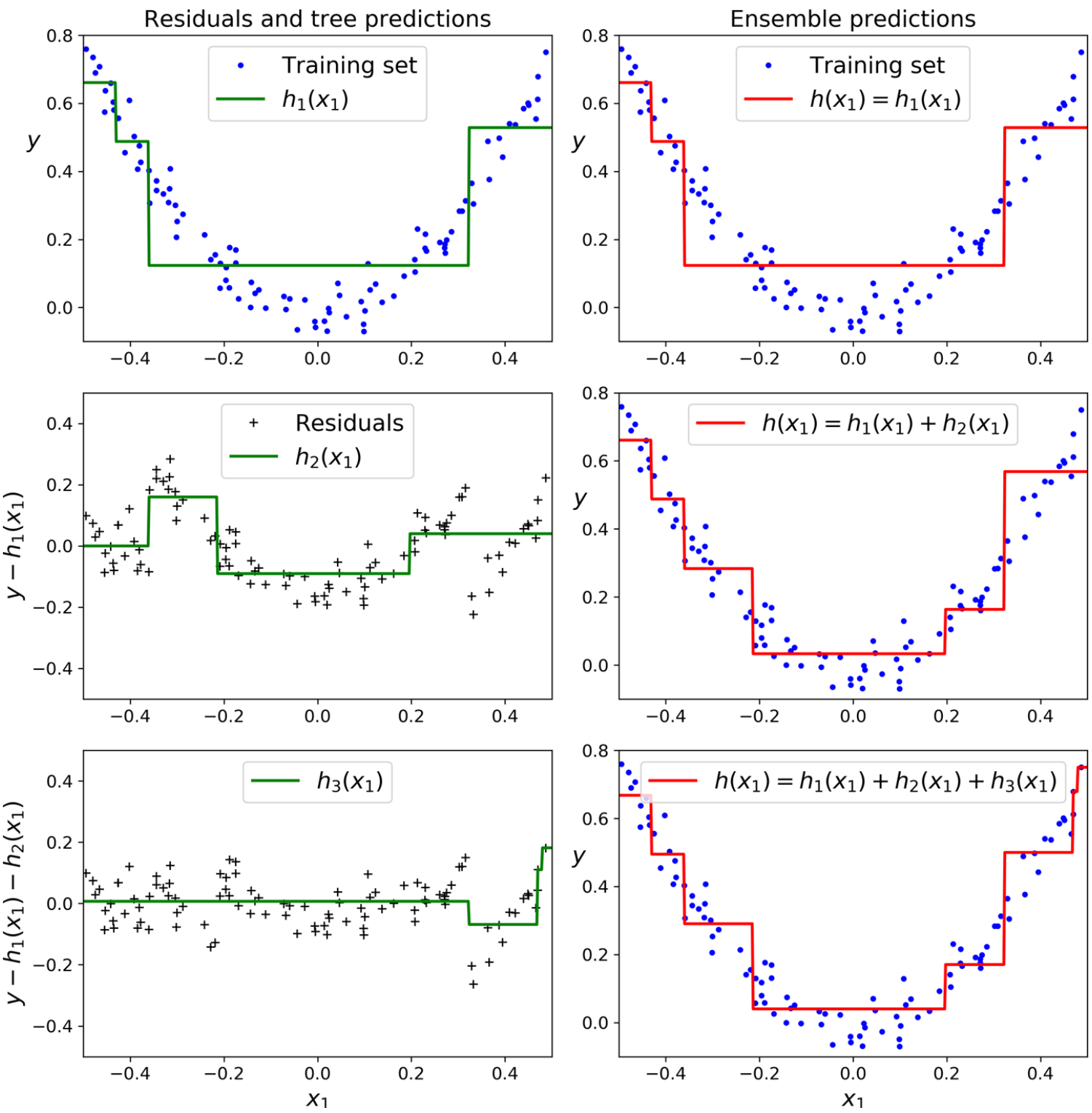


Figure: Gradient Boosting. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

In the above figures on Gradient Boosting, the first predictor (top left) is trained normally, then each consecutive predictor (middle left and lower left) is trained on the previous predictor's residuals; the right column shows the resulting ensemble's predictions (Géron, 2019).

In the first row above, the ensemble has just one tree, so its predictions are exactly the same as the first tree's predictions.

In the second row, a new tree is trained on the residual errors of the first tree. On the right, you can see that the ensemble's predictions are equal to the sum of the predictions of the first two trees.

Similarly, in the third row, another tree is trained on the residual errors of the second tree. You can see that the ensemble's predictions gradually get better as trees are added to the ensemble.

More Examples

Simple example

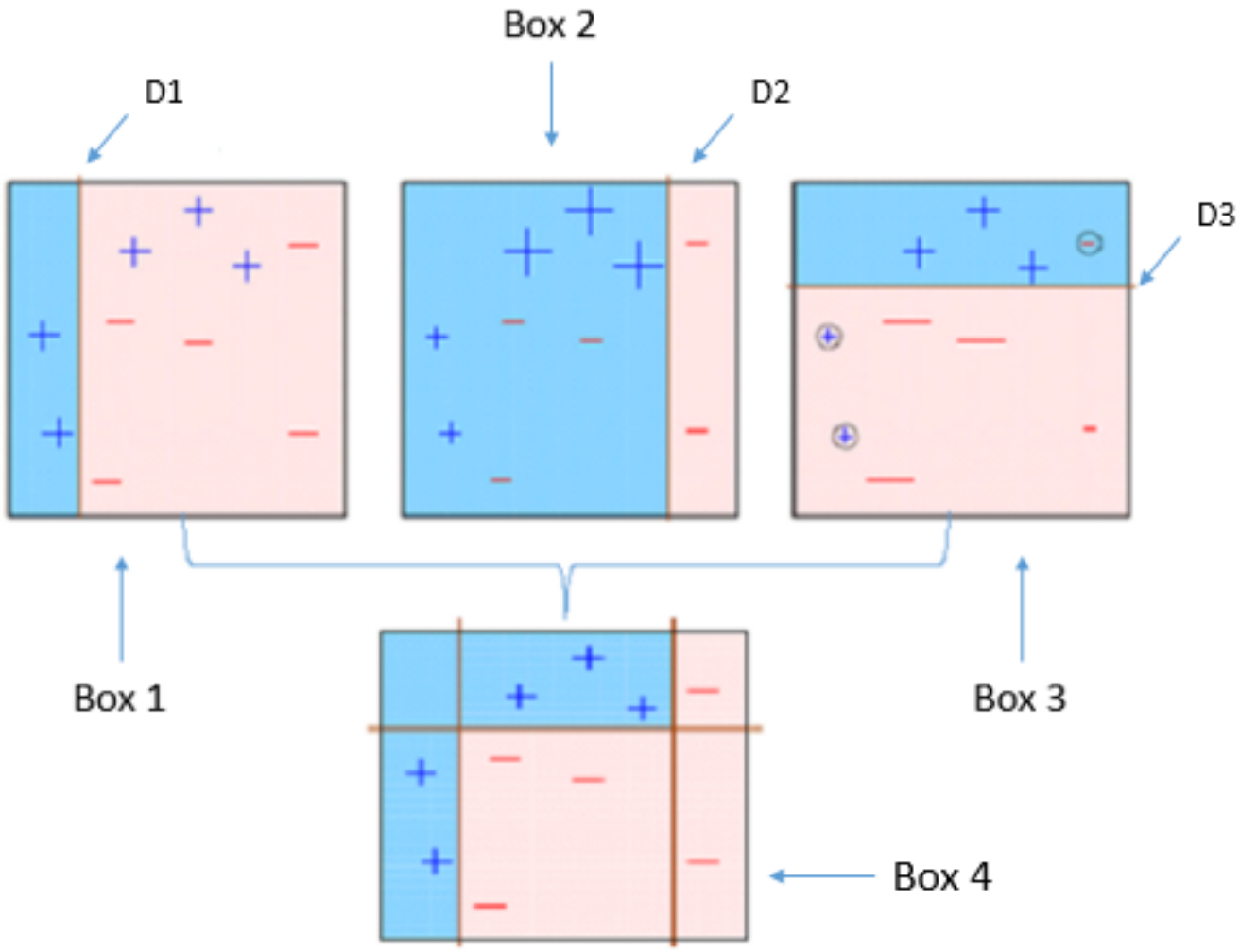


Figure Source. Beginners Tutorial on XGBoost and Parameter Tuning in R: https://www.hackerearth.com/blog/developers/beginners-tutorial-on-xgboost-parameter-tuning-r/

In the above figure, we have four classifiers (4 boxes) that are trying hard to classify `+` and `-` classes as homogeneously as possible.

The details below:

- **Box 1:** The first classifier creates a vertical line (split) at D1. It says anything to the left of D1 is `+` and anything to the right of D1 is `-`. However, this classifier misclassifies three `+` points.
- **Box 2:** The next classifier says don't worry I will correct your mistakes. Therefore, it gives more weight to

the three + misclassified points (see bigger size of +) and creates a vertical line at D2. Again it says, anything to right of D2 is − and left is +. Still, it makes mistakes by incorrectly classifying three − points.

Box 3: The next classifier continues to bestow support. Again, it gives more weight to the three − misclassified points and creates a horizontal line at D3. Still, this classifier fails to classify the points (in circle) correctly.

Note that each of these classifiers has a misclassification error associated with them. Boxes 1,2, and 3 are weak classifiers. These classifiers will now be used to create a strong classifier Box 4.

Box 4: It is a weighted combination of the weak classifiers. As you can see, it does good job at classifying all the points correctly.

Source. Beginners Tutorial on XGBoost and Parameter Tuning in R: https://www.hackerearth.com/blog/developers/beginners-tutorial-on-xgboost-parameter-tuning-r/

Boston Housing Data Example

We select a regression problem which is the prominent Boston Housing dataset. We begin with an approximation; since it is a regression problem with the outcome (MEDV), we take the average of MEDV (AVG) and calculate the residuals of each of the target variable against the average as shown below.

CRIM	NOX	RM	MEDV	AVG	Residuals
			24	26	-2
			21	26	-5
			34	26	8
			33	26	7
			36	26	10
			28	26	2
			22	26	-4
			28	26	2
			16	26	-10
..	..	..	..	..	..

Figure Source. Ensemble methods: https://ajaytech.co/python-ensemble-methods/

Next, instead of fitting the target variable, we fit the residuals. This is almost like doing a linear regression with Gradient Descent, except we are using decision trees (not stumps) as shown below.

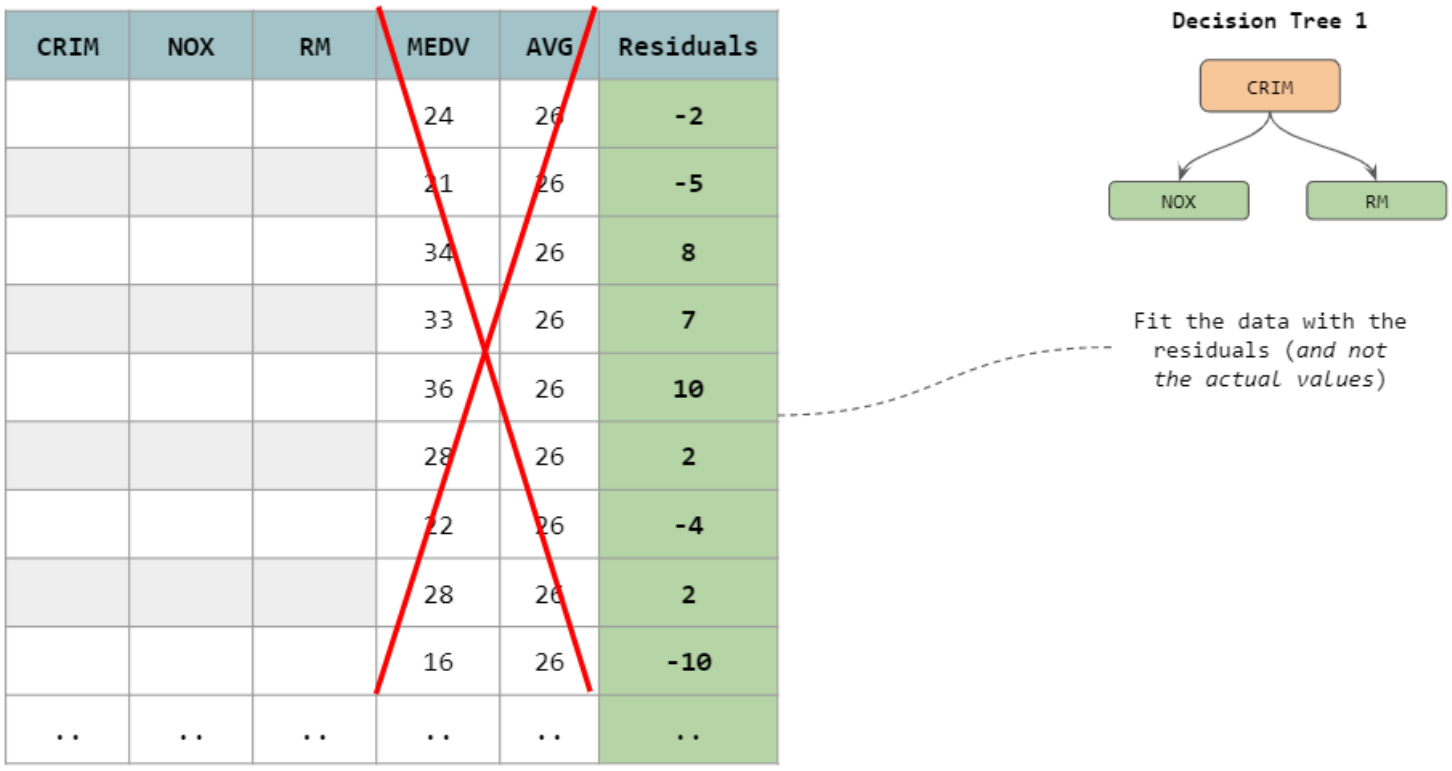


Figure Source. Ensemble methods: https://ajaytech.co/python-ensemble-methods/

Note that we consider a user-chosen **learning rate** (0.1) that is used to compute the prediction (25.8) by taking into account the decision of the tree leaf (-2) and average (26) as shown below.

CRIM	NOX	RM	MEDV	AVG	Residuals	Pred
			24	26	-2	
			21	26	-5	
			34	26	8	
			33	26	7	
			36	26	10	
			28	26	2	
			22	26	-4	
			28	26	2	
			16	26	-10	
..	..	..	..	..	..	

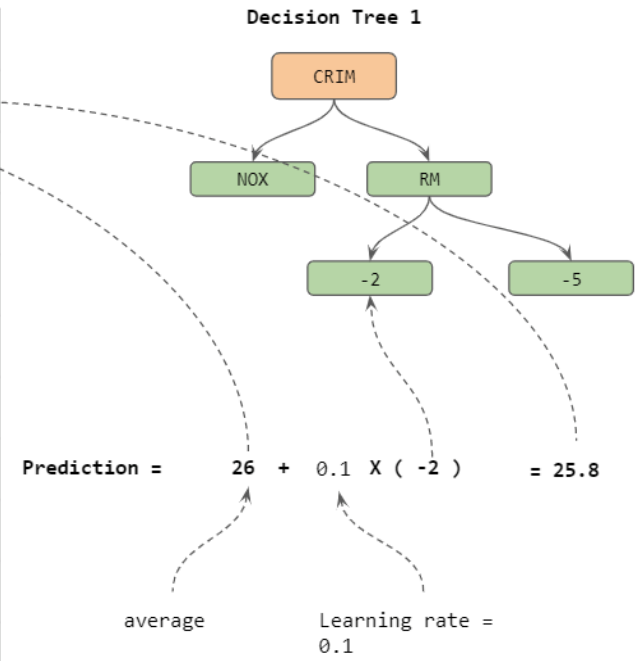


Figure Source. Ensemble methods: https://ajaytech.co/python-ensemble-methods/

The process is continued for the rest of the instances and the prediction is computed as shown below.

CRIM	NOX	RM	MEDV	AVG	Residuals	Pred
			24	26	-2	25.8
			21	26	-5	21.1
			34	26	8	26.08
			33	26	7	26.07
			36	26	10	27
			28	26	2	26.2
			22	26	-4	25.6
			28	26	2	26.2
			16	26	-10	25
..	..	..	..	..	..	

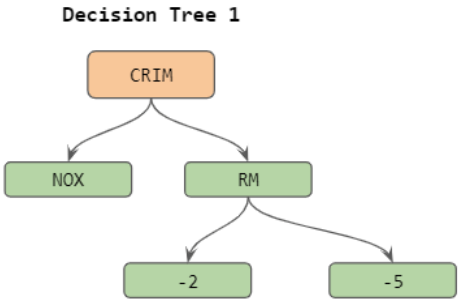


Figure Source. Ensemble methods: https://ajaytech.co/python-ensemble-methods/

Next, we use the new prediction to compute the residuals as shown below. We can see a trend getting the prediction moving towards the actual value and the residuals slowly decreasing.

CRIM	NOX	RM	MEDV	Pred	Residuals	Pred
			24	25.8	-1.8	
			21	21.1	-0.1	
			34	26.8	7.2	
			33	26.7	6.3	
			36	27	9	
			28	26.2	1.8	
			22	25.6	-3.6	
			28	26.2	1.8	
			16	25	-9	

Prev. Pred	Prev. Residuals
26	-2
26	-5
26	8
26	7
26	10
26	2
26	-4
26	2
26	-10

Figure Source. Ensemble methods: https://ajaytech.co/python-ensemble-methods/

Algorithm

Gradient boosting was inspired by the idea that boosting can be interpreted as an optimization algorithm on a suitable cost function.

Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, *26*(3), 801-849: https://projecteuclid.org/download/pdf_1/euclid.aos/1024691079

Explicit regression gradient boosting algorithms were subsequently with the more general functional gradient boosting perspective.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, *38*(4), 367-378:https://doi.org/10.1016/S0167-9473(01)00065-2

Mason, L., Baxter, J., Bartlett, P. L., & Frean, M. R. (2000). Boosting algorithms as gradient descent. In *Advances in neural information processing systems* (pp. 512-518). http://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent.pdf

The above two papers introduced the abstract view of boosting algorithms as iterative functional gradient descent algorithms where algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. The functional gradient view of boosting has led to the application of boosting algorithms in many areas of machine learning and statistics

Friedman's Gradient Boosting Algorithm for a generic loss function is given by $L(y_i, \gamma)$ and the algorithm is shown below. Note that y_i and $\gamma)$ are actual and predicted values, respectively. L is the loss and to minimise the loss, we need to take first-order derivative as given below.

$$L(y_i, \gamma) = (y_i - \gamma)^2$$

$$\frac{\delta L}{\delta \gamma} = -2(y_i - \gamma)$$

where

$$f(x_i) = \gamma$$

Below see the algorithm for M trees in the ensemble, where we compute r_{im} residuals in (a) for N instances (denoted by i) and M models (denoted by m). Note in part (d), although not explicitly shown, a user-chosen **learning rate** ν in range of [0,1] is used to multiply each tree output in the ensemble for summation in case of a regression problem.

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ 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}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

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

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

Source: Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Source: Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

You can run the following code segment to display individual predictions and the derived prediction value of the ensemble method (Gradient Boosting). Change the value of `X_new` and re-run.

Run PYTHON

```
import numpy as np
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)

```

We can implement the above using the class `GradientBoostingRegressor` as shown below.

Click on 'Run'.

Run

PYTHON

```
import numpy as np
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3,
                                 learning_rate=1.0, random_state=42)
model = gbrt.fit(X, y)

X_new = np.array([[0.8]])
print('gbrt.predict : ', gbrt.predict(X_new) )

```

Learn about learning rates and apply them using Python.

Learning rates

Often new models in Gradient Boosting quickly fit the training data set, resulting in overfit. We can slow down the learning by applying a weighting factor for the corrections by new predictors. This weighting is called the **learning rate** or **shrinkage**. A learning rate of less than 1.0 results in more trees required to be added to the ensemble, offering better generalisation. Normally learning rates are low in the range of 0.1 to 0.3.

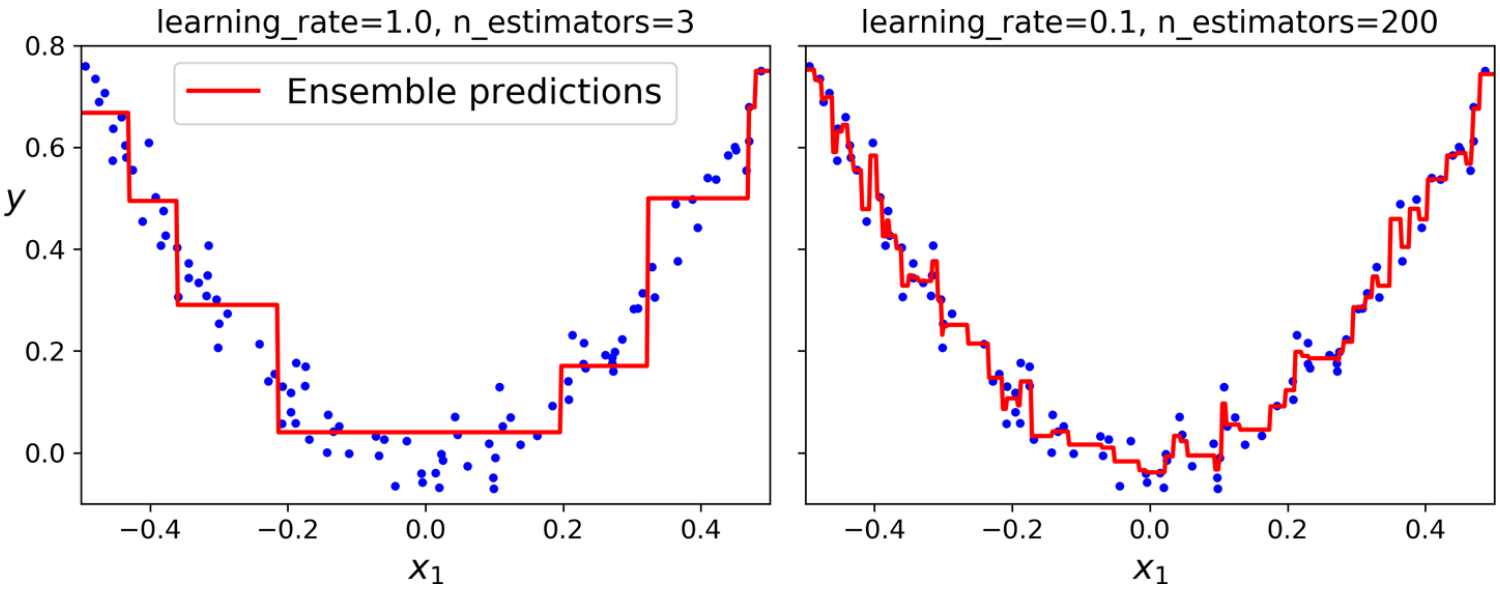


Figure: GBRT ensembles with not enough predictors (left with learning rate 1.0) and too many (right with learning rate 0.1). Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Predict y-values for x=0.36 when `learning_rate` is 1.0 and later 0.1.

Run PYTHON

```python
import numpy as np
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3,
                                 learning_rate=1.0, random_state=42)
model = gbrt.fit(X, y)

X_new = np.array([[0.36]])
print('gbrt.predict : ', gbrt.predict(X_new) )

```

We next show Scikit-learn implementation that highlights the effect of different regularization strategies for Gradient Boosting.

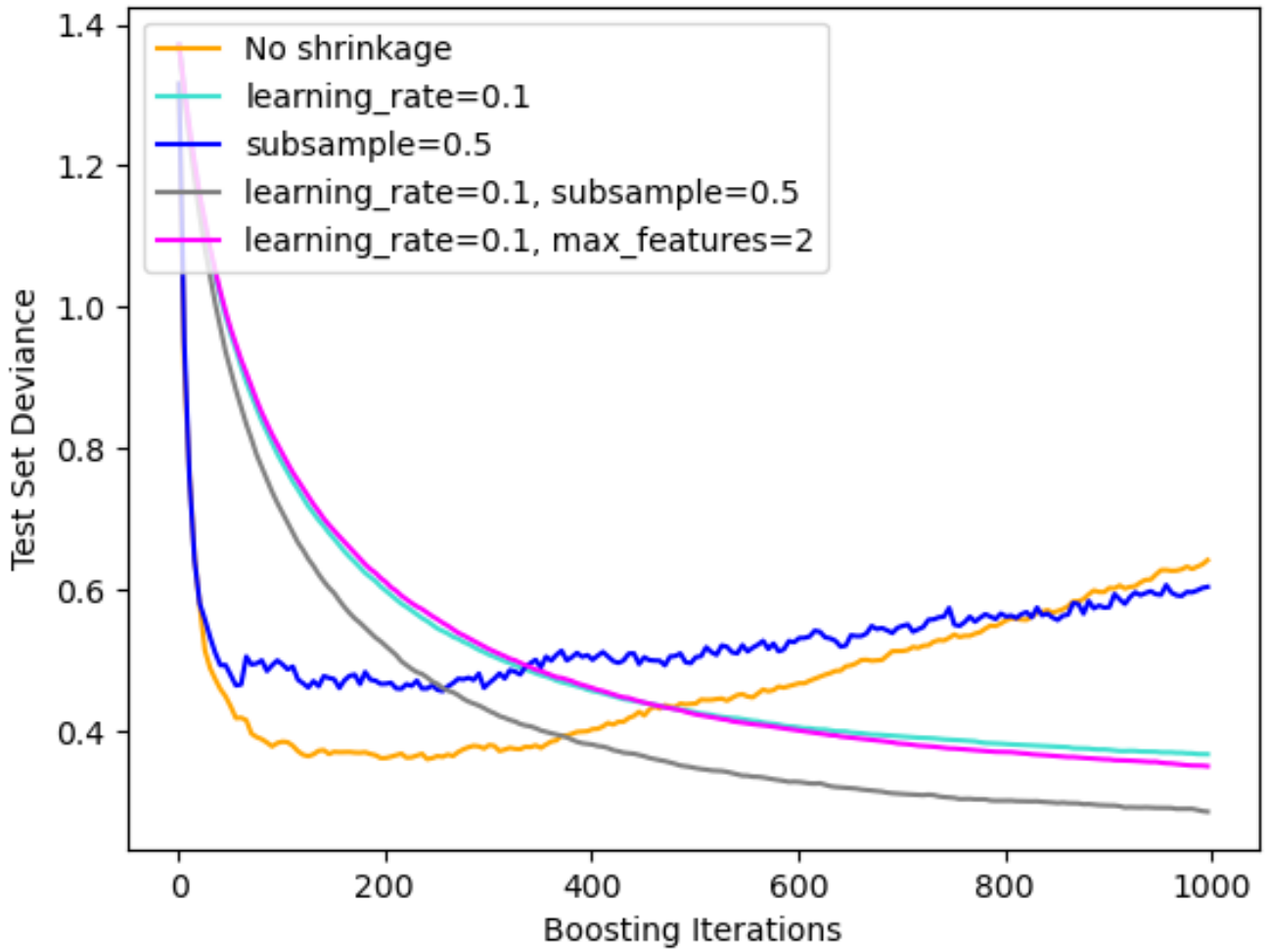
The loss function used is binomial deviance and we find that regularization via shrinkage (`learning_rate < 1.0`) improves performance considerably. In combination with shrinkage, stochastic gradient boosting (`subsample < 1.0`) can produce more accurate models by reducing the variance via bagging. The subsampling without shrinkage has shown poor trend. The code and results are shown below.

Run PYTHON

```python
import numpy as np
import matplotlib.pyplot as plt

from sklearn import ensemble
from sklearn import datasets


X, y = datasets.make_hastie_10_2(n_samples=12000, random_state=1)
X = X.astype(np.float32)

# map labels from {-1, 1} to {0, 1}
labels, y = np.unique(y, return_inverse=True)

X_train, X_test = X[:2000], X[2000:]
```

Code Source: https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regularization.html#sphx-glr-auto-examples-ensemble-plot-gradient-boosting-regularization-py

The output is shown below.



Next we see R implementation

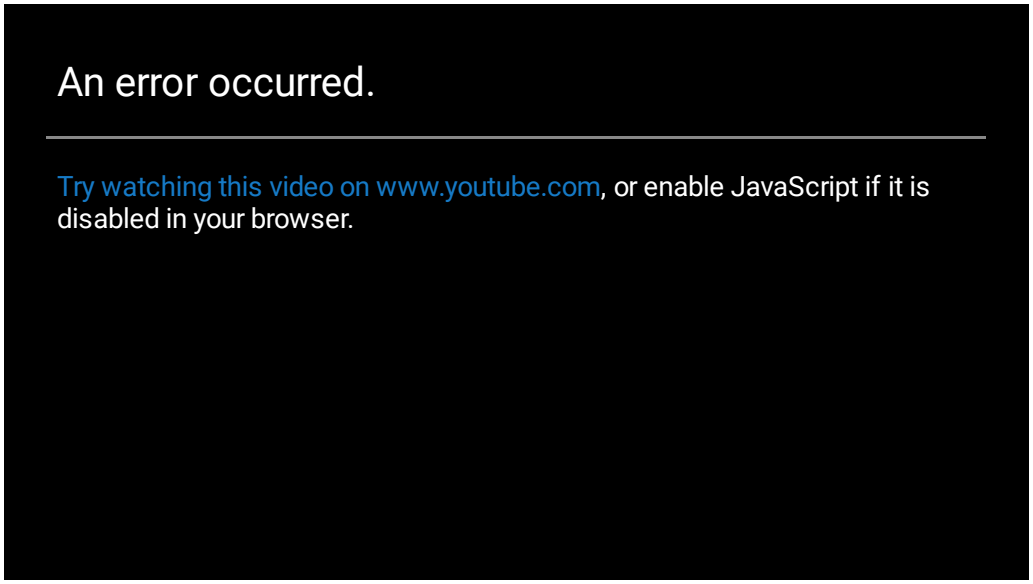
Run R

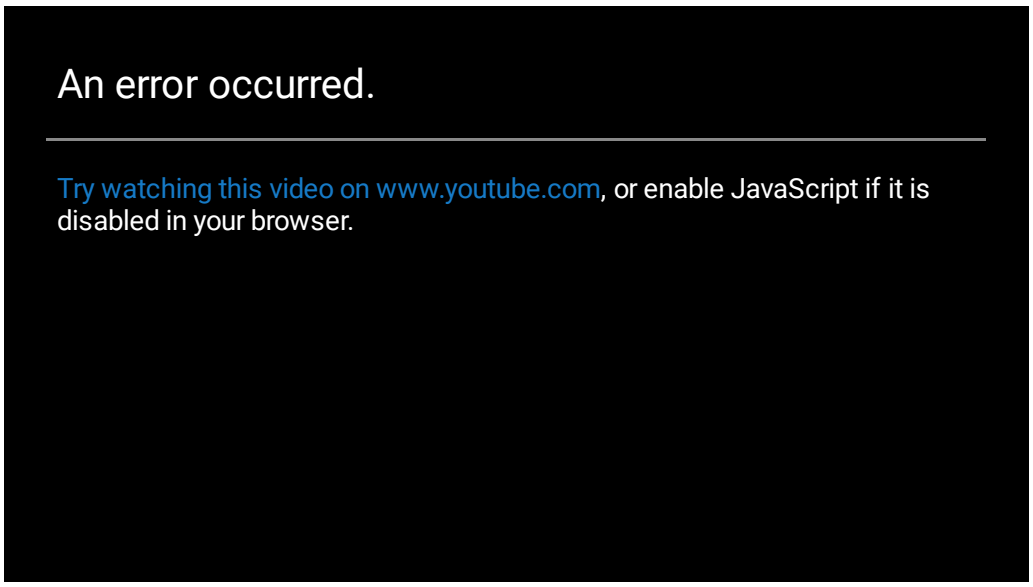
```r
library(gbm)
library(pROC)
#library(cvAUC)
# Load 2-class HIGGS dataset
train <- data.table::fread("https://s3.amazonaws.com/erin-data/higgs/hig
test <- data.table::fread("https://s3.amazonaws.com/erin-data/higgs/higg


#print(train)  large dataset - around 10K instances

set.seed(1)
model <- gbm(
  formula = response ~ .,
  distribution = "bernoulli",
```

Run R

```r
#source: http://uc-r.github.io/gbm_regression# packages ----------------
library(rsample)
library(caret)
library(ggthemes)
library(scales)
#library(wesanderson)
library(tidyverse)
library(gbm)
#library(Metrics)
library(here)

#https://www.storybench.org/tidytuesday-bike-rentals-part-2-modeling-wit
```

The videos below summarize the method.





References

1. Breiman, L. (1996). *Bias, variance, and arcing classifiers*. Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA: https://www.stat.berkeley.edu/users/breiman/arcall96.pdf
2. Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, *26*(3), 801-849: https://projecteuclid.org/download/pdf_1/euclid.aos/1024691079
3. Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, *38*(4), 367-378:https://doi.org/10.1016/S0167-9473(01)00065-2
4. Mason, L., Baxter, J., Bartlett, P. L., & Frean, M. R. (2000). Boosting algorithms as gradient descent. In *Advances in neural information processing systems* (pp. 512-518). http://papers.nips.cc/paper/1766-boosting-algorithms-as-gradient-descent.pdf
5. Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, *7*, 21: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full

Exercise 1

Using Dataset from Project I Case I, and Abalone Dataset from Project II Option II, with R or Python

1. Provide average classification and prediction performance for 10 experimental runs using Adaboost and Gradient Boosting Machine.
2. Compare the results with Adam/SGD and Random Forests and Decision Trees.
3. Highlight difference in performance and what do you see regarding the standard decision of the 10 experimental runs. Are ensemble methods more certain with their predictions? (i.e is standard deviation lower than neural networks?). Discuss this issue.

XGBoost

XGBoost follows the principle of gradient boosting with a difference in modelling details. XGBoost uses a more regularized model formalization to control over-fitting, which gives it better performance.

XGboost actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms; however, the more appropriate name could be regularized gradient boosting.

XGBoost is a popular Python library offering optimised implementation of Gradient Boosting with advantages below. R implementation is also available.

XGBoost Advantages

- **Regularization:** Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.
- **Parallel Processing:** XGBoost implements parallel processing and is **blazingly faster** as compared to GBM. We know that boosting is a sequential process so how can it be parallelized? We know that each tree can be built only after the previous one, so what stops us from making a tree using all cores?
- **High Flexibility:** XGBoost allows users to define **custom optimization objectives and evaluation criteria**. This adds a whole new dimension to the model and there is no limit to what we can do.
- **Handling Missing Values:** XGBoost has an in-built routine to handle missing values. The user is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
- **Tree Pruning:** A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a **greedy algorithm**. 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. Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.
- **Built-in Cross-Validation:** XGBoost 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. This is unlike GBM where we have to run a grid-search and only a limited values can be tested.
- **Continue on Existing Model:** User can start training an XGBoost model from its last iteration of previous

run. This can be of significant advantage in certain specific applications. GBM implementation of sklearn also has this feature so they are even on this point.

Source: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

XGBoost Parameters

We must set three types of parameters: general parameters, booster parameters and task parameters.

- **General parameters** relate to which booster we are using to do boosting, commonly tree or linear model
- **Booster parameters** depend on which booster you have chosen
- **Learning task parameters** decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.
- **Command line parameters** relate to behavior of CLI version of XGBoost.

Random Forests vs Boosted Trees: The difference arises from how we train them. One uses bagging and the other uses boosting which can be in form of AdaBoost or Gradient Boosting or XGBoost. In Random Forest, the way the bag or element for the ensemble is created is by randomly shooing set of instances and attributes while in Boosted Trees, the data is created depending on the requirements. AdaBoost creates a new set of data while Gradient Boosting and XGBoost works with the same dataset in terms of features but trains on a different set of outcomes, i.e residuals.

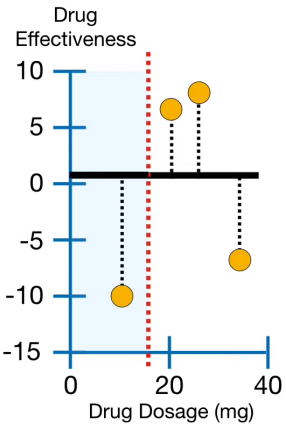
Example

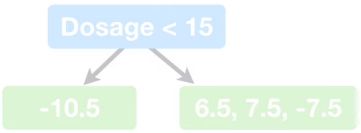
In the next set of figures, we demonstrate the issue of pruning the trees in XGBoost ensemble which considers Gain (which is a measure of information quality), γ which is user-defined parameter that determines when we need to prune and λ which is regularisation parameter.

We first make a tree based on data.

Predicted Drug Effectiveness

0.5





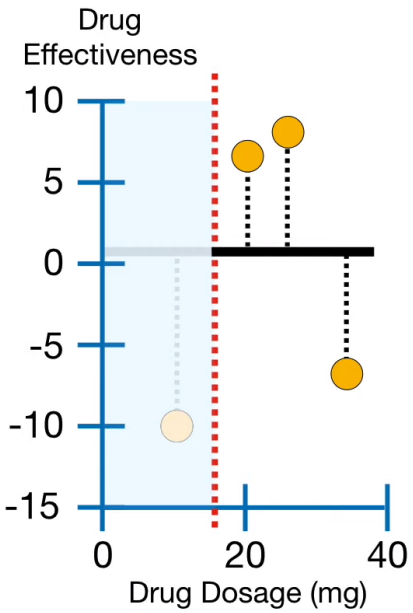
$$\text{Similarity Score} = \frac{\text{Sum of Residuals, Squared}}{\text{Number of Residuals} + 1}$$

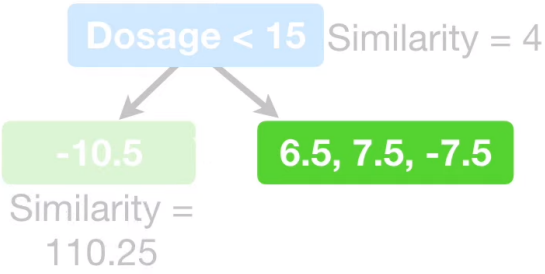
Remember λ (lambda) is a **Regularization Parameter**, which means that it is intended to reduce the prediction's sensitivity to individual observations.

We calculate the Gain of the Node by first looking at the Similarity Score.

Predicted Drug Effectiveness

0.5





$$\text{Similarity Score} = \frac{(6.5 + 7.5 + -7.5)^2}{\text{Number of Residuals} + \lambda}$$

…and since there are **3 Residuals** in the leaf on the right, we plug **3** into the denominator…

Then we calculate the Gain

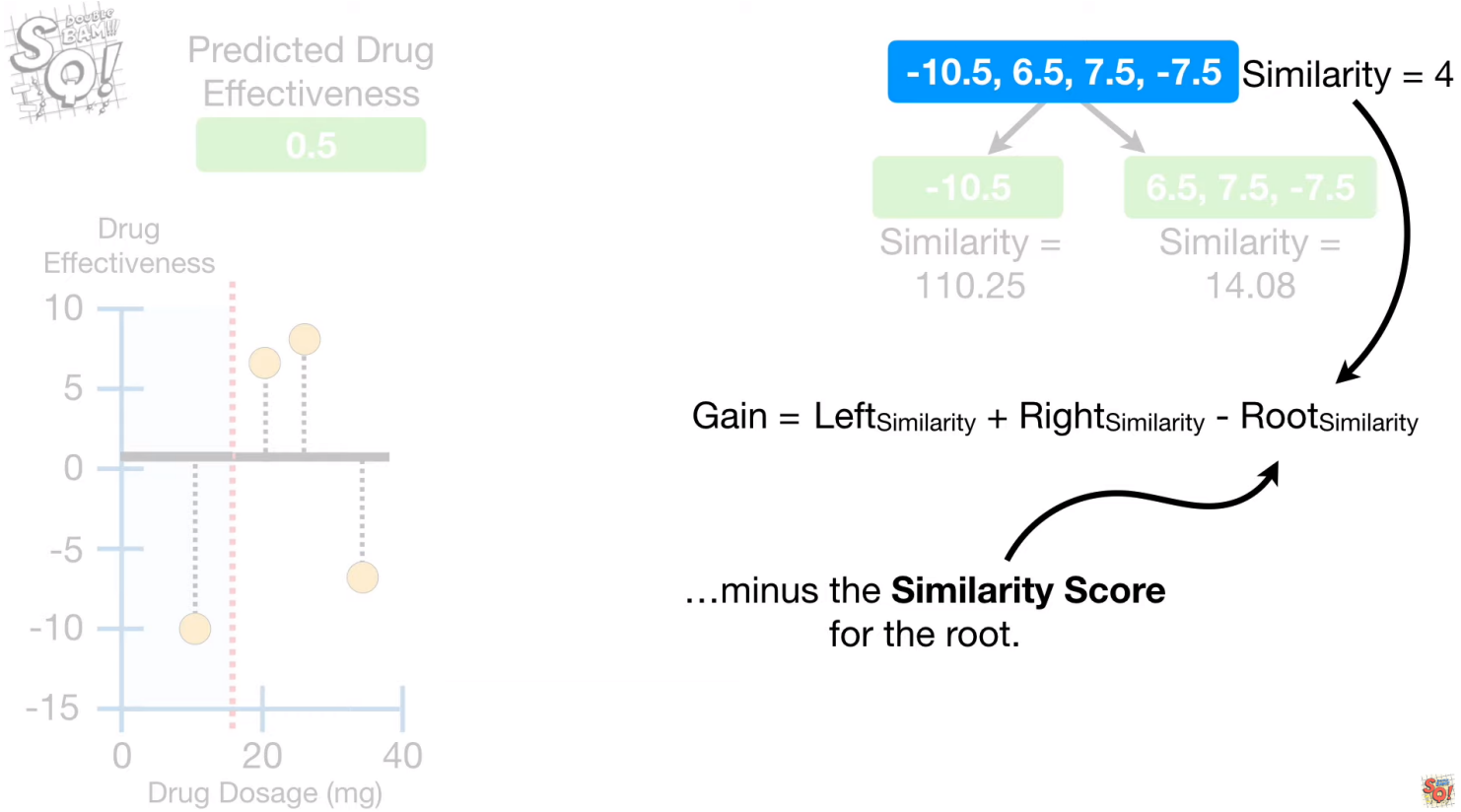


Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E&feature=emb_logo

We consider if the tree needs to be pruned.

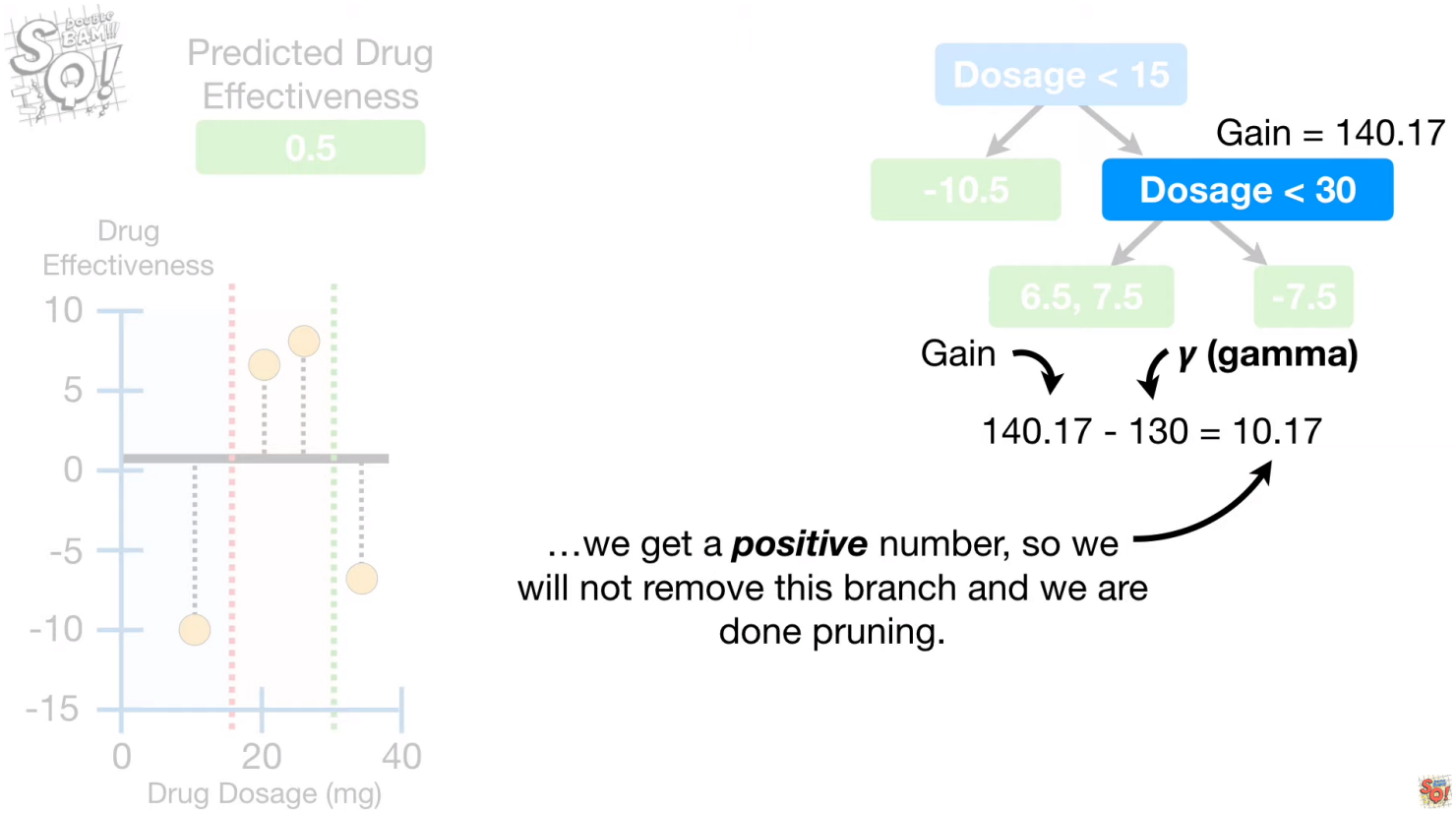


Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E&feature=emb_logo

In the above figure, we decide not to remove Dosage < 30 since the outcome of Gain - gamma is a positive number.

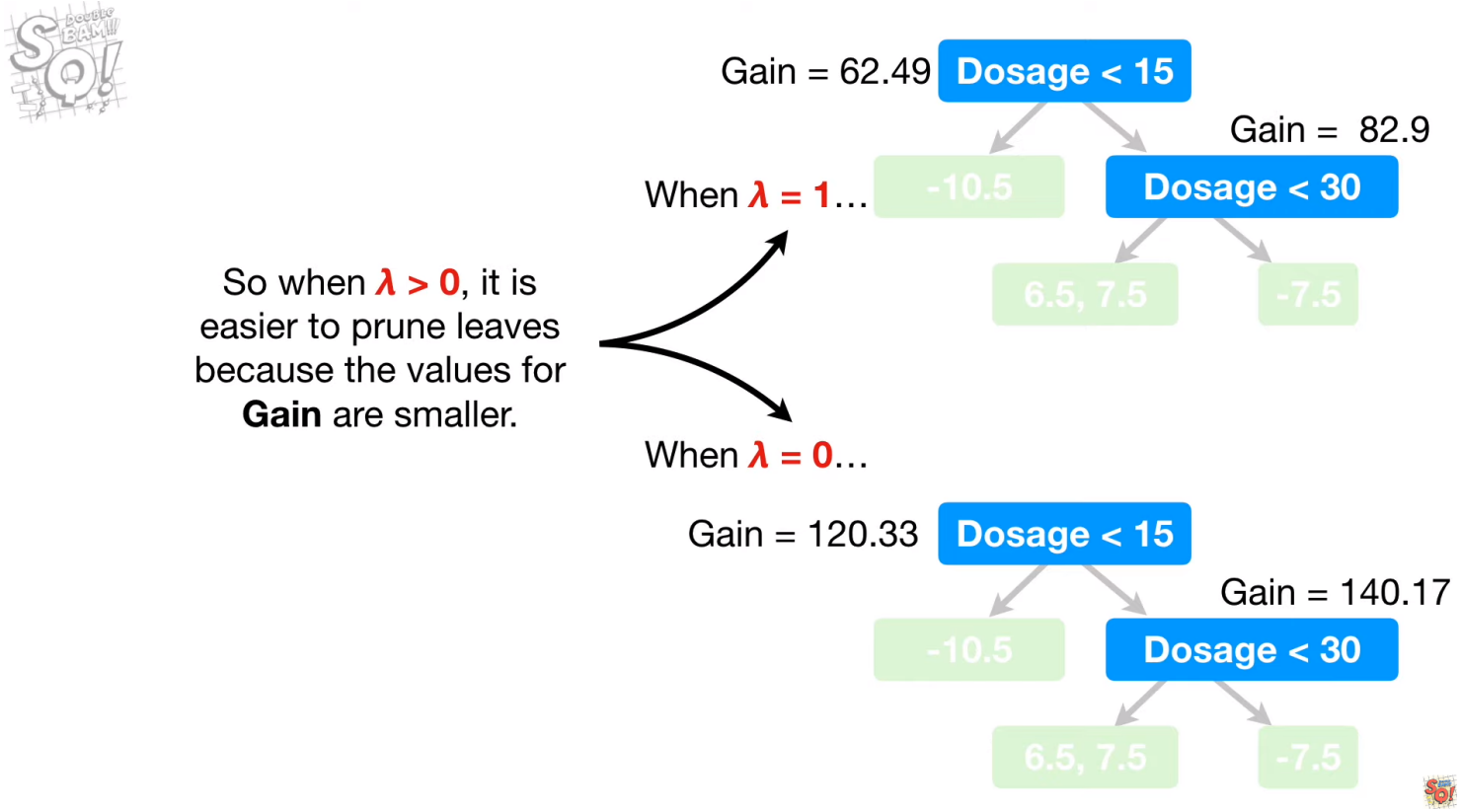
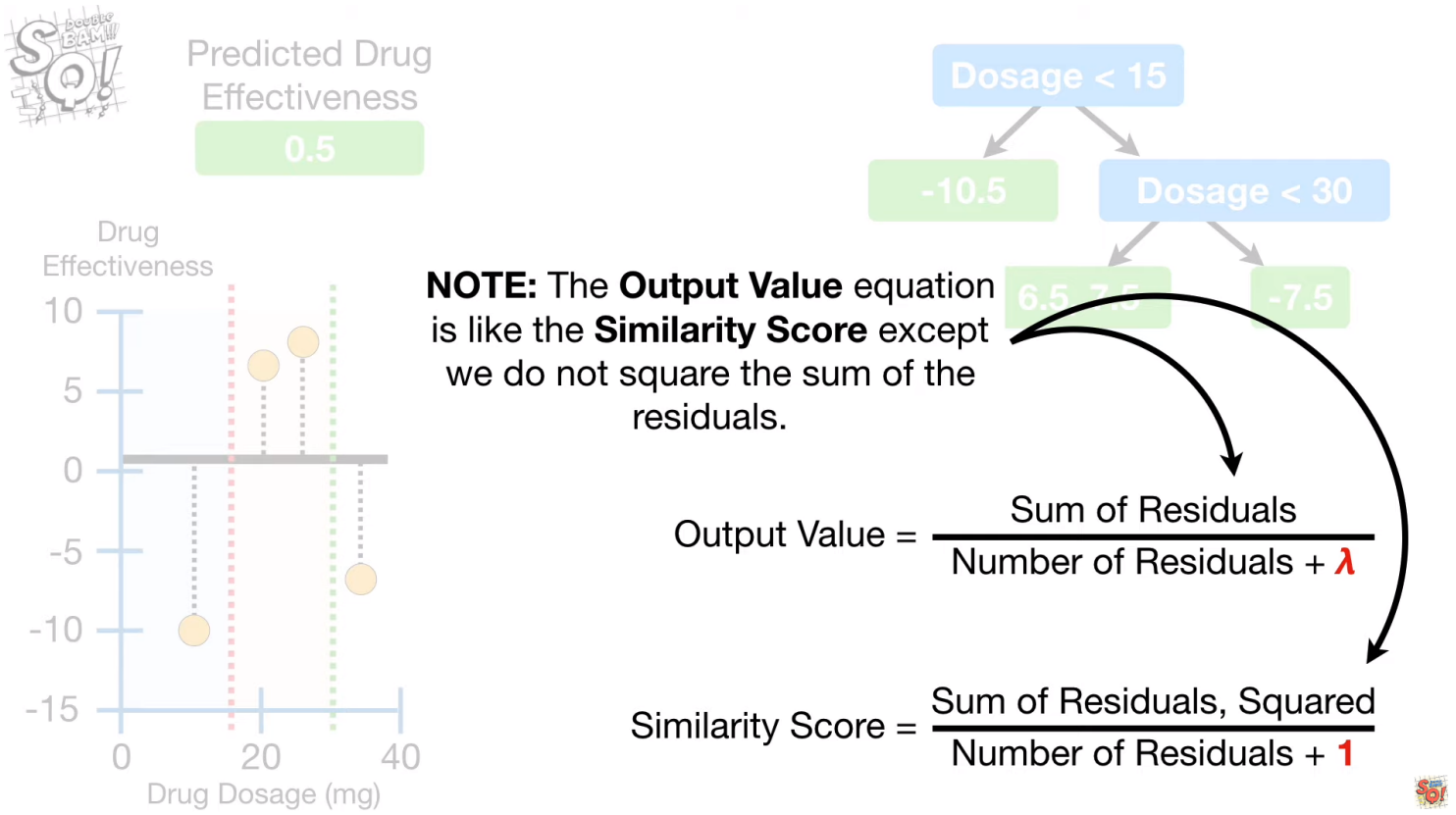


Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E&feature=emb_logo

Next, we can create another tree based on previous residuals and then try to prune by using Gain and gamma.



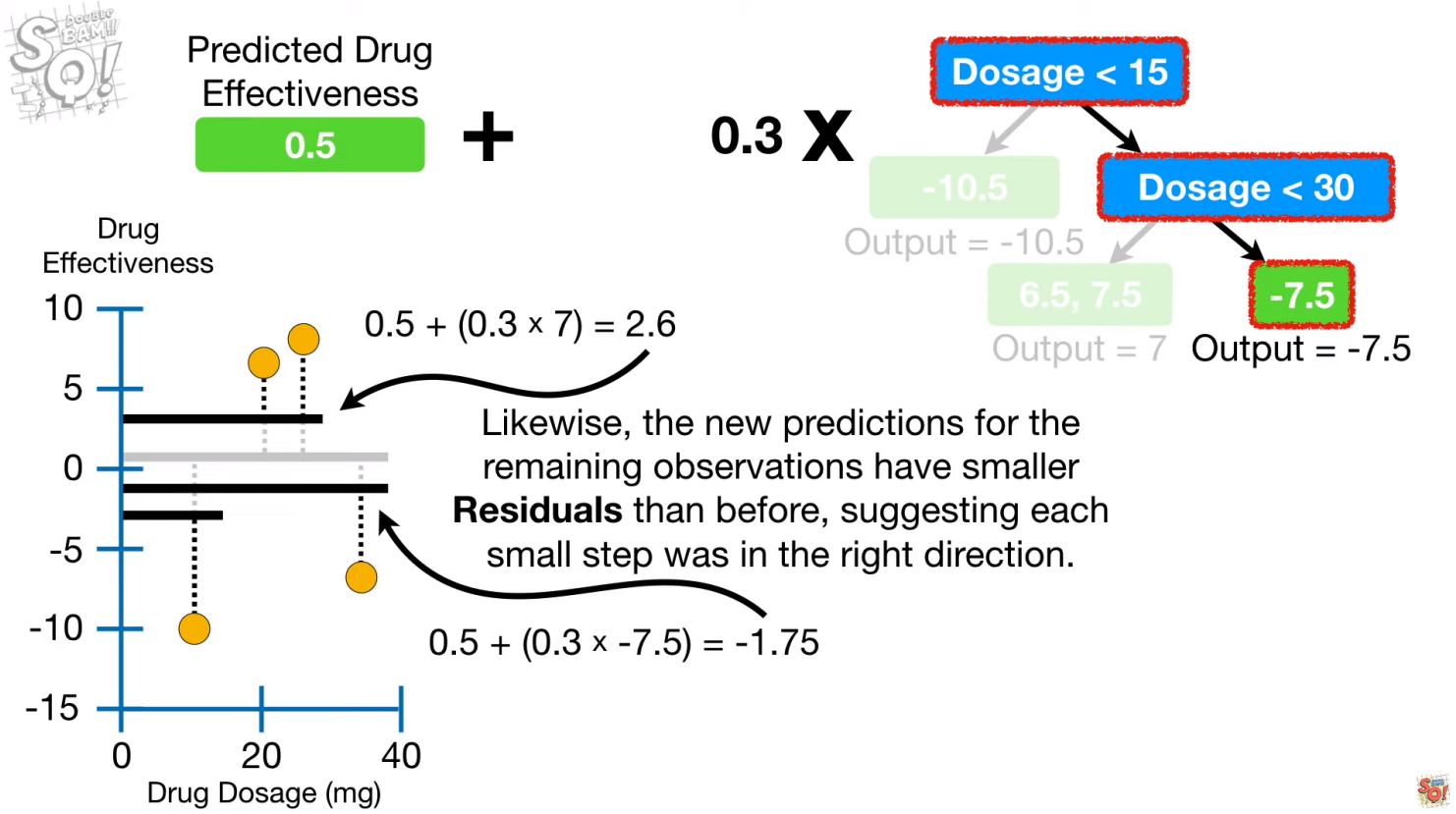


Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E&feature=emb_logo

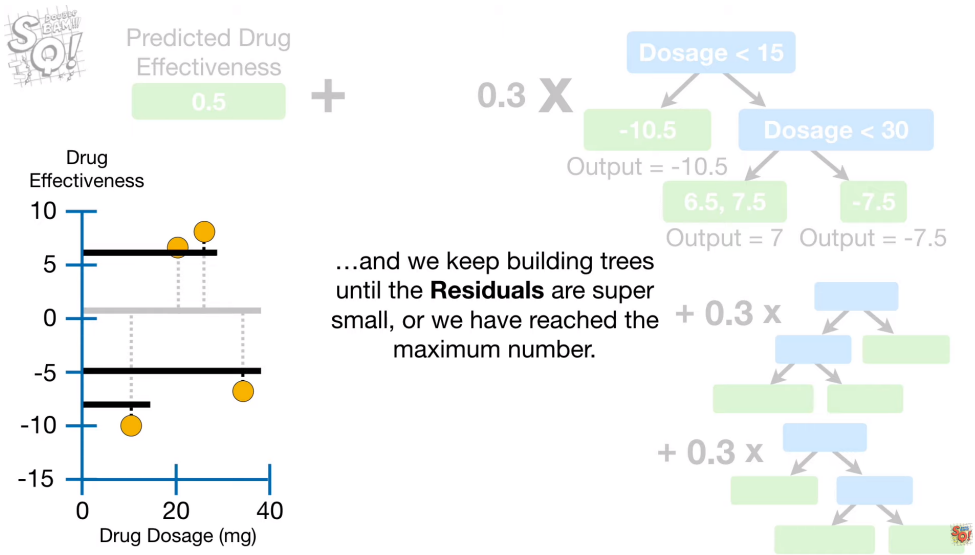
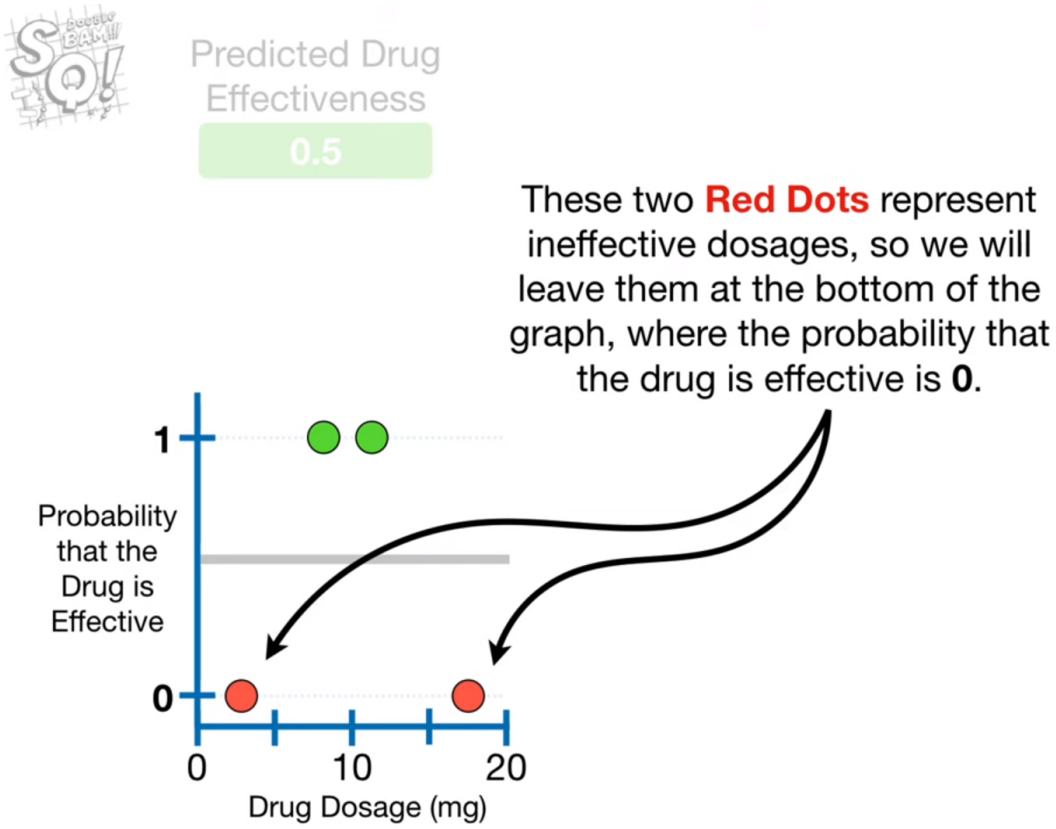


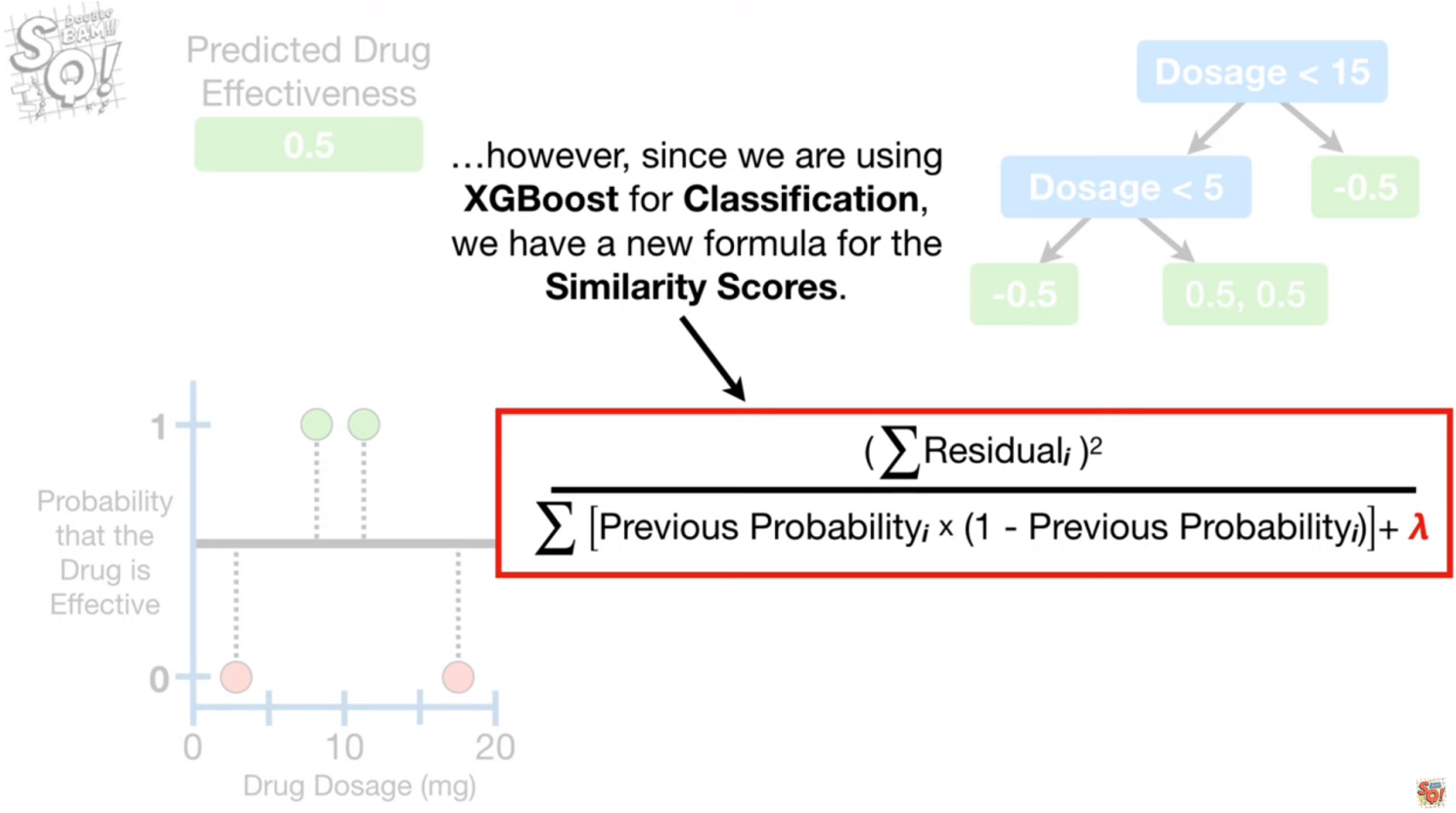
Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E&feature=emb_logo

The process repeats for the selected number of trees in the ensemble and the final collective decision is made.

And for classification, we follow similar pattern.



With key difference in similarity score.



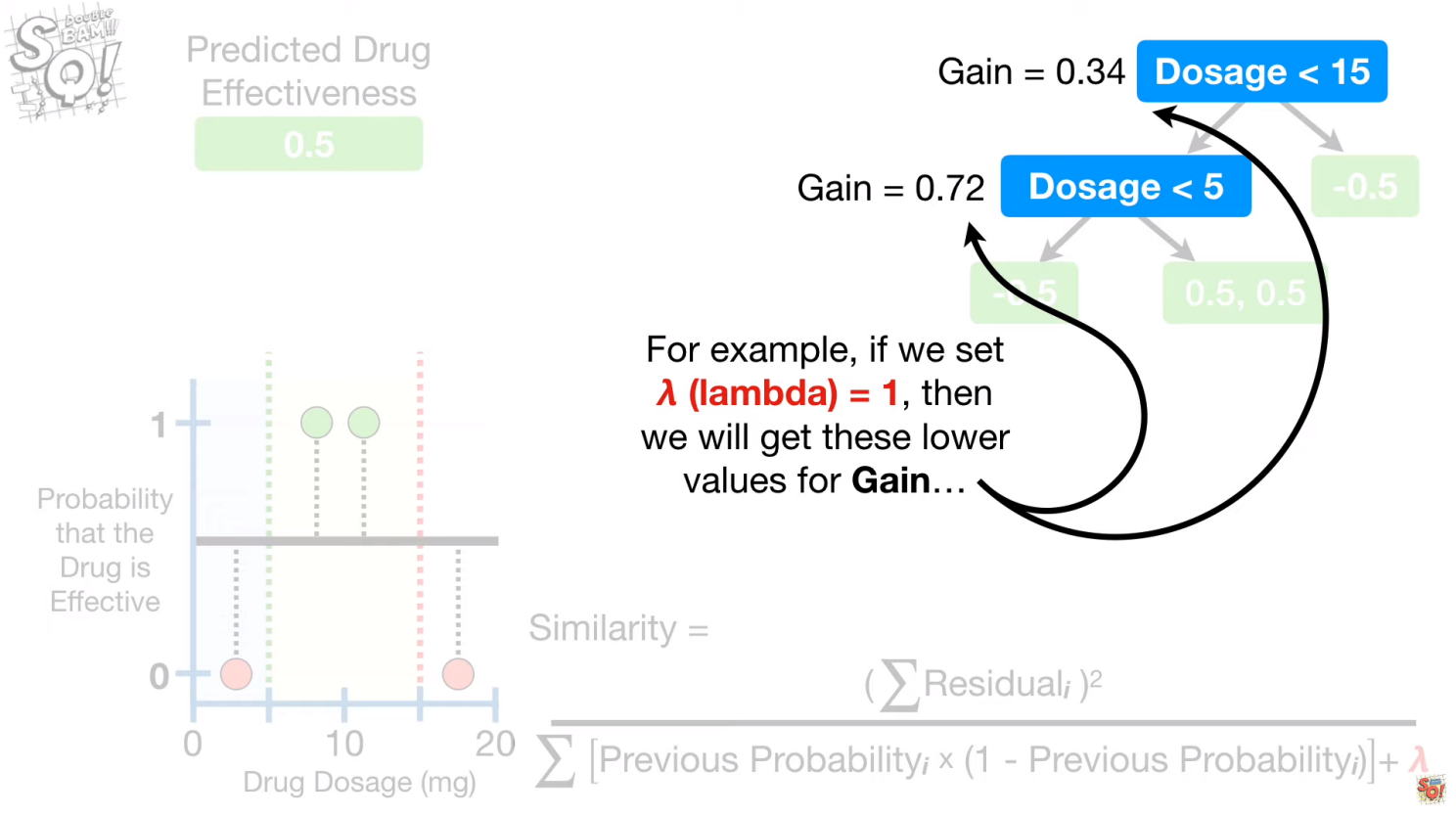


Figure Source. XGBoost Part 1: Regression: https://www.youtube.com/watch?v=8b1JEDvenQU

Algorithm

The objective function consist of two parts: **training loss** and **regularization term**:

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

where L is the training loss function, and Ω is the regularization term which controls the complexity of the model, which helps us to avoid overfitting.

Let us consider the following problem in the following picture where we are asked to *fit* visually a step function given the input data points on the upper left corner of the image. Which solution among the three do you think is the best fit?

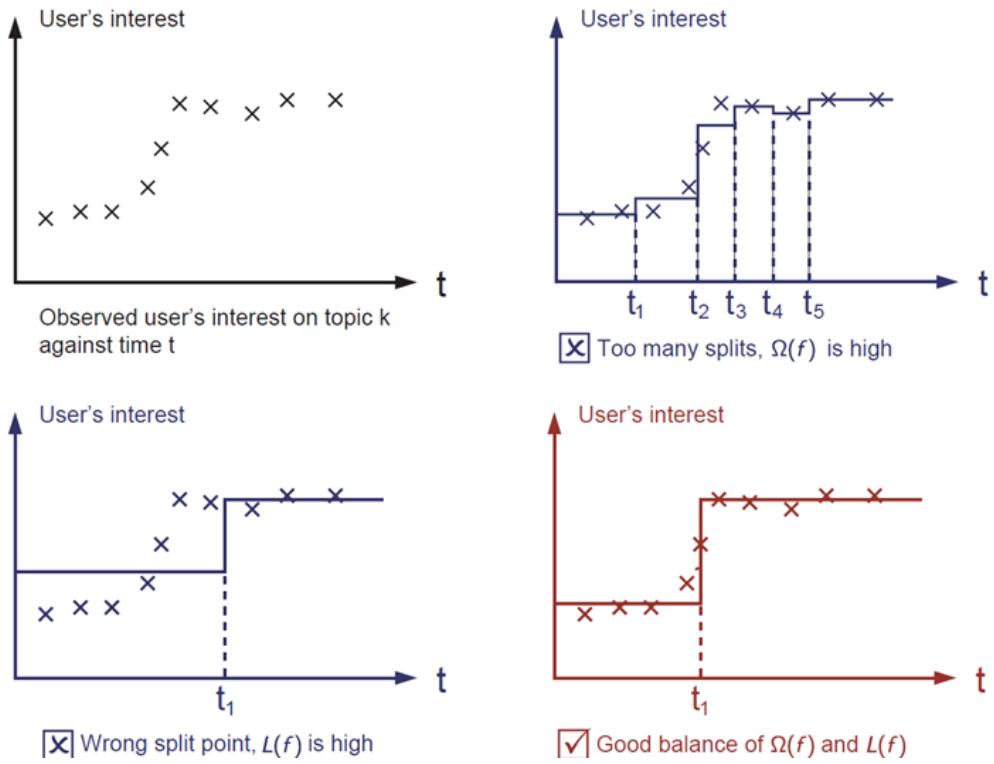


Figure Source. Introduction to Boosted Trees: https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Below is a simple example of a CART that classifies whether someone will like a hypothetical computer game X.

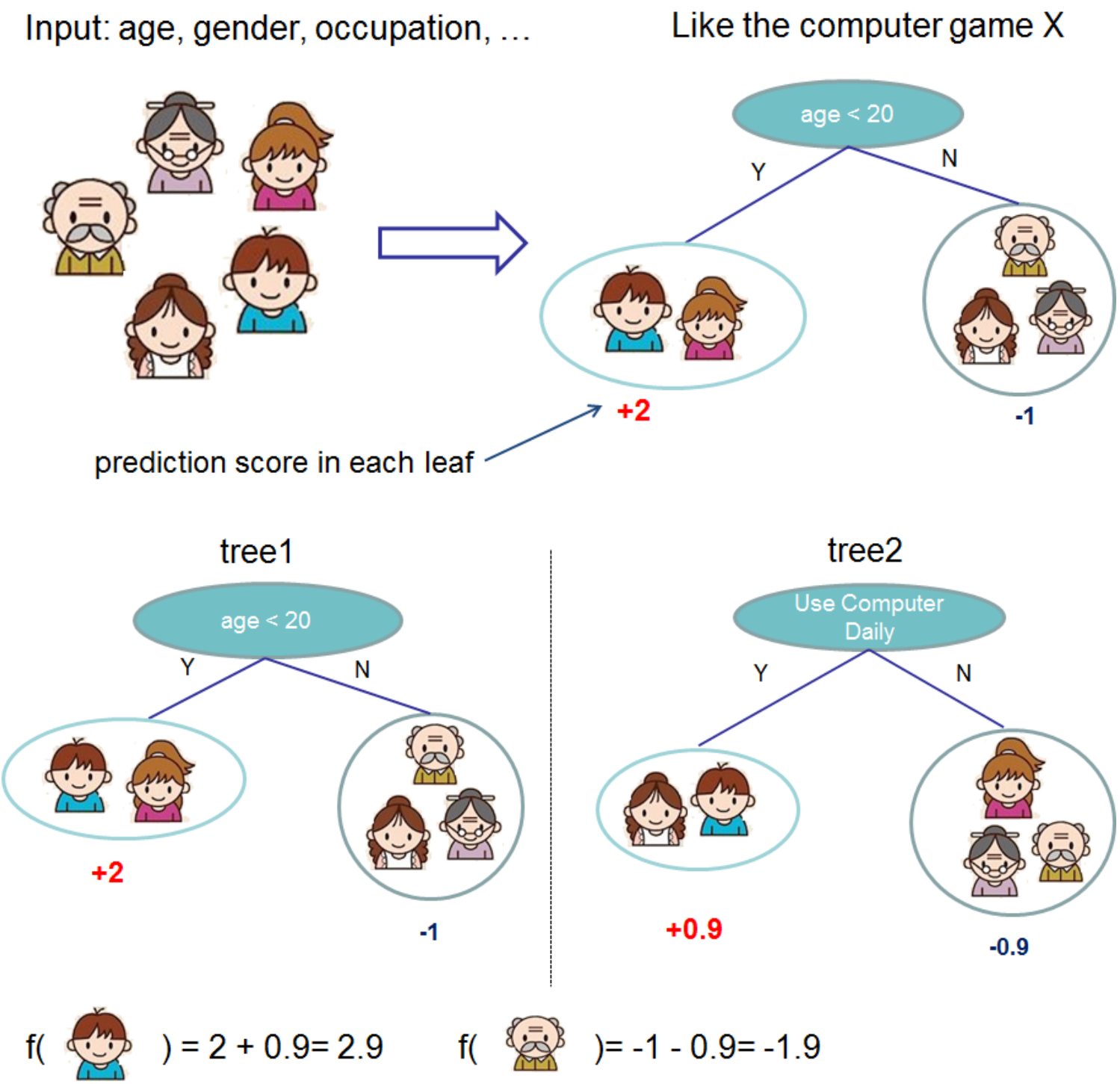


Figure Source. Introduction to Boosted Trees: https://xgboost.readthedocs.io/en/latest/tutorials/model.html

We classify the members of a family into different leaves, and assign them the score on the corresponding leaf. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification when compared to conventional decision trees.

XGBosst uses the same Gradient Boosting ensemble model, which sums the prediction of multiple trees together. The major **difference is that there is regularisation** as the trees are constructed via pruning.

Below code in Scikit-learn:

Run PYTHON

```python
from sklearn.datasets import load_boston
boston = load_boston()
#The boston variable itself is a dictionary, so you can check for its ke

print(boston.keys())
#dict_keys(['data', 'target', 'feature_names', 'DESCR'])
#You can easily check for its shape by using the boston.data.shape attri

print(boston.data.shape)
#(506, 13)
#As you can see it returned (506, 13), that means there are 506 rows of

import pandas as pd

```

Code Source. Using XGBoost in Python :https://www.datacamp.com/community/tutorials/xgboost-in-python

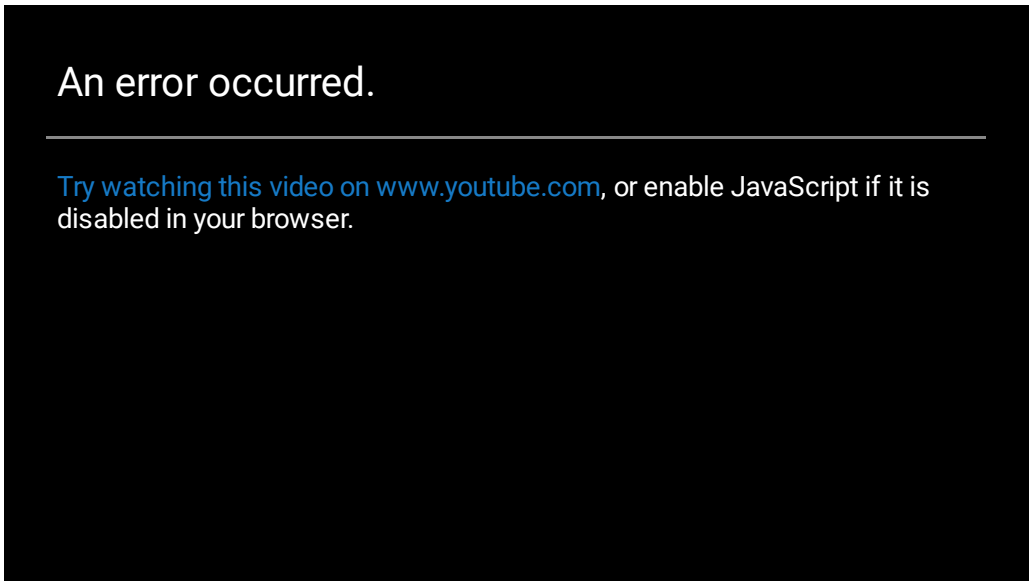
As you can see the feature `RM` has been given the highest importance score among all the features. Thus XGBoost also gives you a way to do Feature Selection.

Run R

```r

#Source: https://koalaverse.github.io/machine-learning-in-R/gradient-boo
library(xgboost) # Ed to install
library(Matrix)
#library(cvAUC) # Ed to install
# convert data.table to data frames
#train <- as.data.frame(train)
#test <- as.data.frame(test)


```

Code Source: https://koalaverse.github.io/machine-learning-in-R/gradient-boosting-machines.html

Next we show a video that summarizes the concept.



References

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.
2. Chen, T., He, T., Benesty, M., Khotilovich, V., & Tang, Y. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1-4: http://cran.fhcrc.org/web/packages/xgboost/vignettes/xgboost.pdf
3. Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794): https://dl.acm.org/doi/pdf/10.1145/2939672.2939785?casa_token=MHNhvpFNDL8AAAAA:PkwXyuprbxtF-yGifwglZcmRbqossbF7GE5qeGxhla3IiDelysC7IVrLumACq7-5Op93tAo62l7A4HE

Stacking

Stacked generalisation or **stacking** uses meta learner instead of voting (or an average) to combine predictions of base learners. Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model). If base learners can output probabilities it's better to use those as input to meta learner. It offers more information to meta learner.

In principle, any learning scheme can be applied for meta learner; however, it is preferable to use "relatively global, smooth" models (Wolpert, 1992). Since base learners do most of the work, this avoids the risk of overfitting. The following figure shows a meta-learner (blender) making a final prediction (3.0) based on the prediction values from three level-0 models (3.1, 2.7 and 2.9).

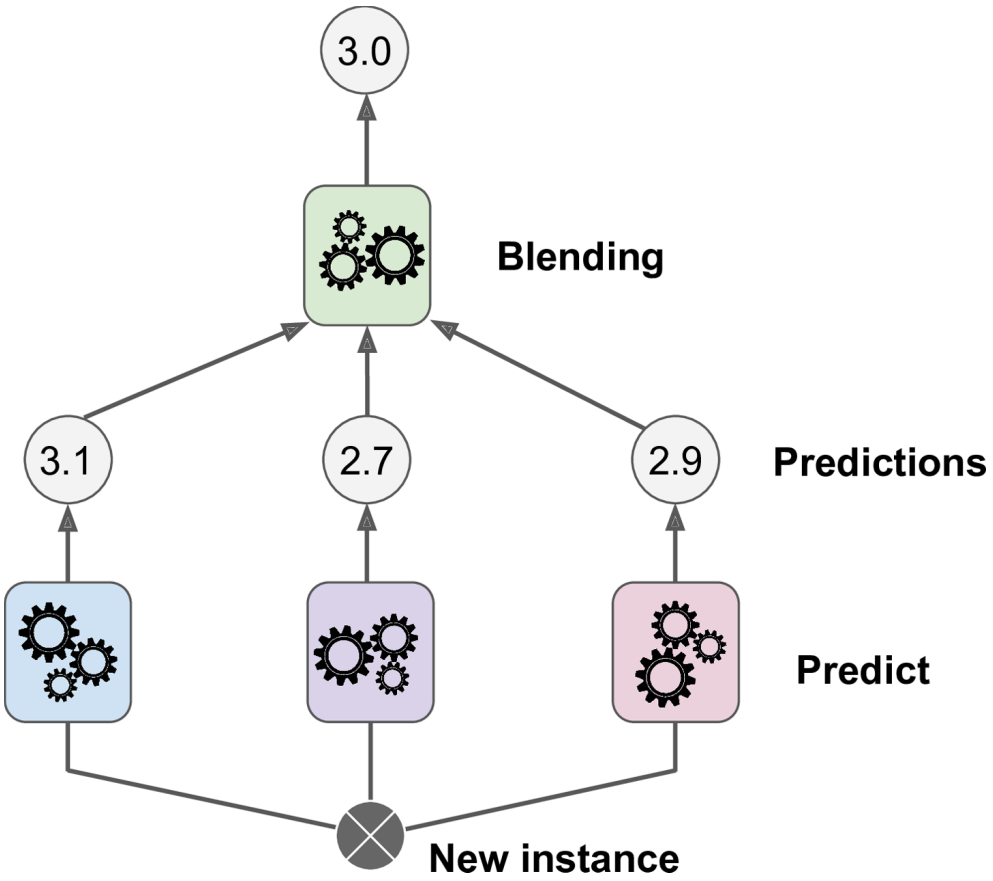


Figure: Aggregating predictions using a blending predictor. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Simply, you define a stack of models for each layer and execute the first model from the first layer all

the way down to the last model in the last layer, while making predictions between levels. These stacks of models could be any type of model, e.g. we could have a stack with XGBoost, Neural Networks and Linear Regression.

The figure below helps in understanding how models are stacked in layers.

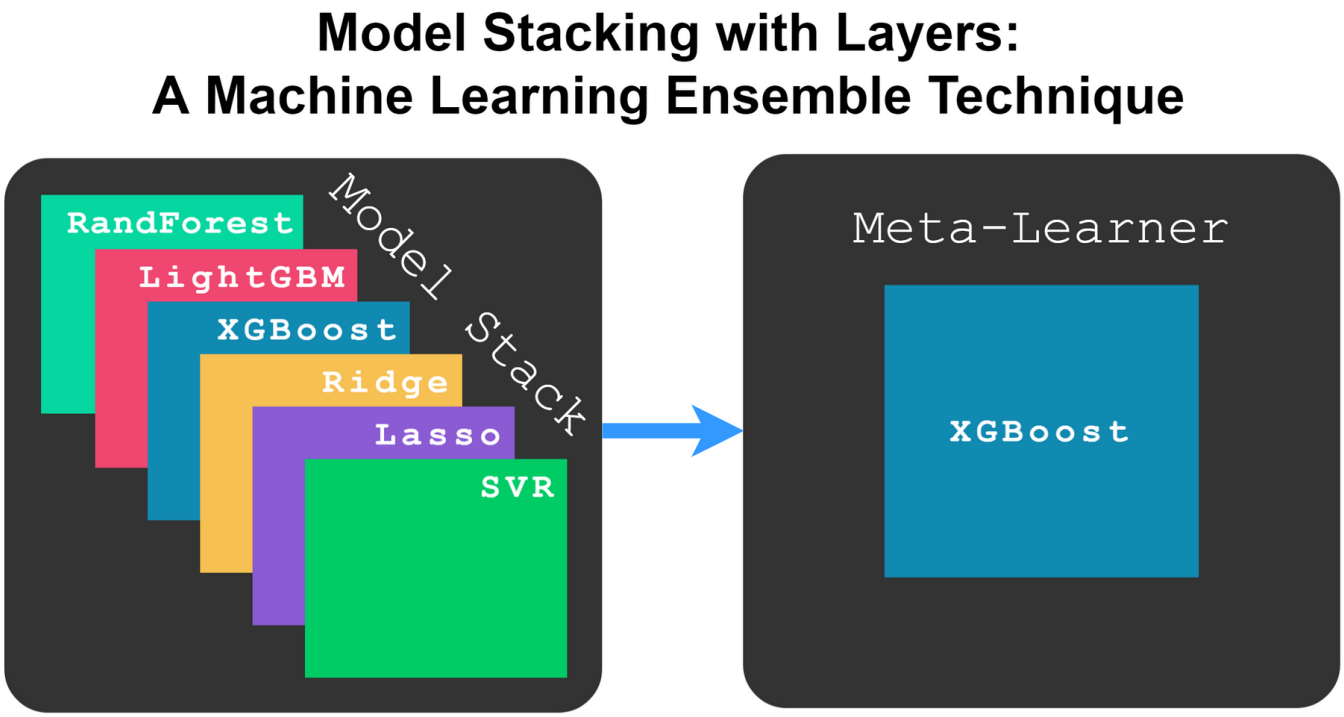


Figure Source. Stack Machine Learning Models - Get Better Results: https://mlfromscratch.com/model-stacking-explained/#/

Advantages of Stacking

- Stacking is popular in data mining competition and a major reason is that it is flexible and simple when compared to other ensemble methods.
- Wolpert argues that model stacking *deduces* the bias in a model on a particular dataset which has been shown in many applications.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2), 241-259. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S089360800580023.

- Bayes Model Averaging (BMA) is similar and has been around before stacking. It has been shown that model stacking with cross-validation out-performs BMA while being robust because model stacking is less sensitive to changes in a dataset.

Clarke, B. (2003). Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. *Journal of Machine Learning Research*, *4*(Oct), 683-712. https://www.jmlr.org/papers/volume4/clarke03a/clarke03a.pdf

Hoeting, J. A., Madigan, D., Raftery, A. E., & Volinsky, C. T. (1999). Bayesian model averaging: a tutorial. *Statistical science*, 382-401: https://www.jstor.org/stable/pdf/2676803.pdf

There is no support for stacking in scikit-learn; however, it is not difficult to quickly create a custom solution. See the figure below which shows how an original dataset is used for K-fold cross-validation. There is an extra step which takes the predictions of each of the folds and uses them as features for further training by another model which is the stacking step for a simple situation.

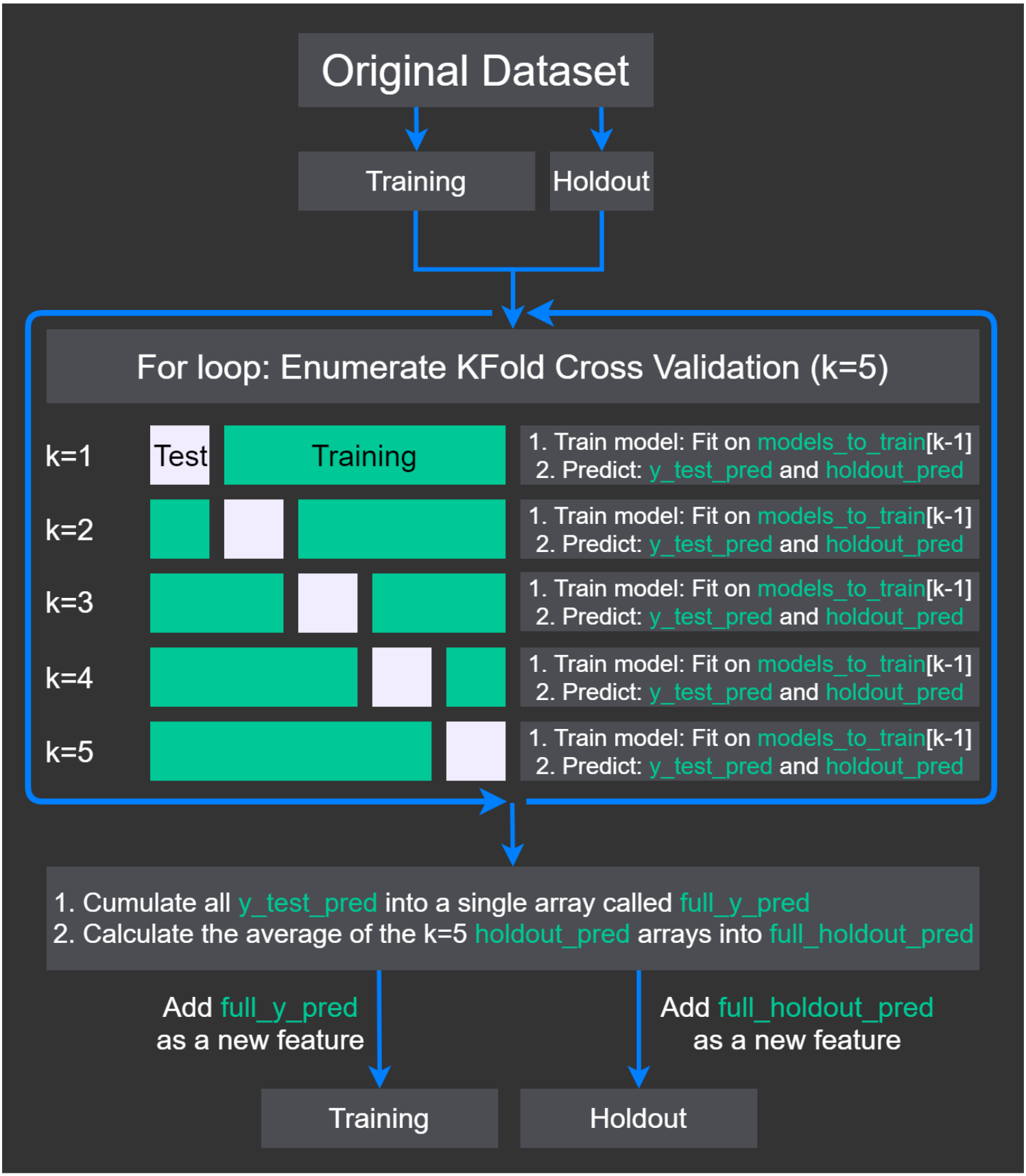
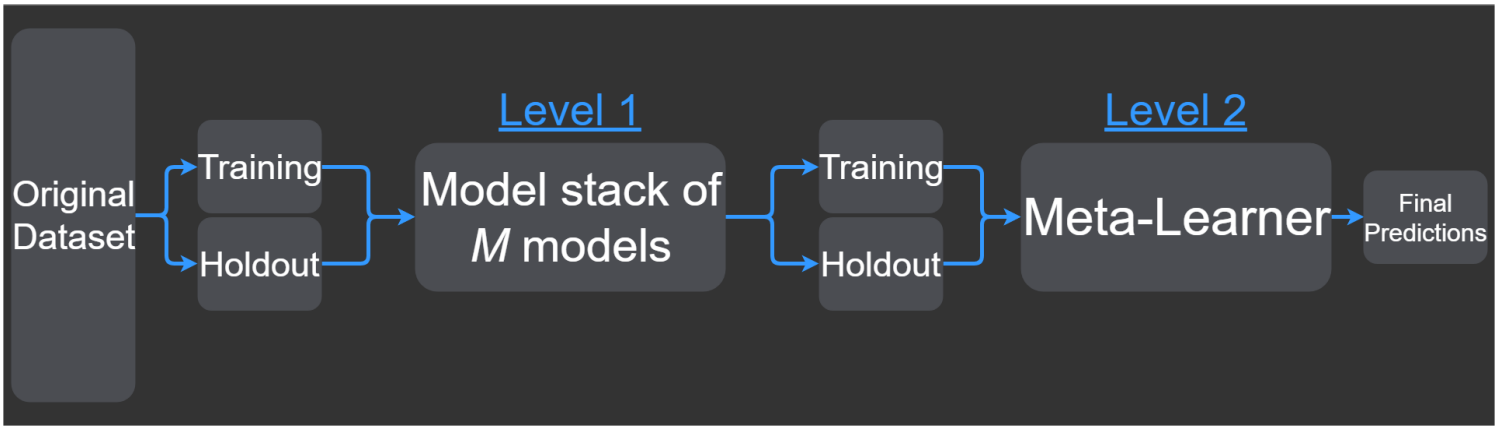


Figure Source. Stack Machine Learning Models - Get Better Results: https://mlfromscratch.com/model-stacking-explained/#/

If you wanted to add more layers, we could just add another level after level 1 and make the meta-learner level 3. If we were to increase the numbers of layers, level 2 would be another round of

model stacking of M models, just as in level 1. The meta-learner is always present in the last layer, hence why we insert a new level before the meta-learner.



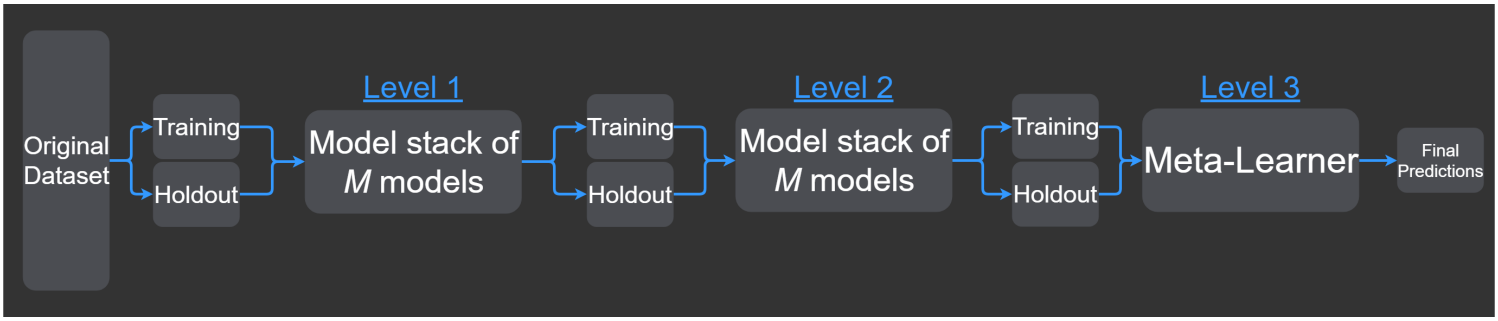


Figure Source. Stack Machine Learning Models - Get Better Results: https://mlfromscratch.com/model-stacking-explained/#/

In summary, we produce new features by different models stack which is combined in a new dataset to make final predictions. At times, other methods can be combined in these model stacks, such as feature selection or normalization.

Now we see how stacking can be done in python using scikit-learn. First, we use a dataset and select several models and see how they perform.

Run PYTHON

```python
# compare standalone models for binary classification
#https://machinelearningmastery.com/stacking-ensemble-machine-learning-w
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot

```

Code Source: https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

Then we use the predictions from the respective models for stacking the models together. It is clear that the stacked model improves the results further.

Run PYTHON

```python
# compare ensemble to each baseline classifier
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
from matplotlib import pyplot

```

Code Source: https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

References

1. Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2), 241-259. Retrieved from https://www.sciencedirect.com/science/article/abs/pii/S089360800580023.
2. Clarke, B. (2003). Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. *Journal of Machine Learning Research*, *4*(Oct), 683-712. https://www.jmlr.org/papers/volume4/clarke03a/clarke03a.pdf

Stacking neural networks

https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/

Run PYTHON

```
import seaborn as sns
import pandas as pd

import matplotlib.pyplot as plt

sns.set_style("white")

# Import data
df = pd.read_csv('https://raw.githubusercontent.com/EarthByte/paleoclima
print(df)


x1 = df.loc[df.cut=='Ideal', 'depth']
x2 = df.loc[df.cut=='Fair', 'depth']
```

Stacking

Apply stacking using Python.

Run the individual classifiers from the previous exercise to make predictions on the validation set, and create a new training set with the resulting predictions: each training instance is a vector containing the set of predictions from all your classifiers for an image, and the target is the image's class.

Train a classifier on this new training set.

Congratulations, you have just trained a blender, and together with the classifiers, it forms a stacking ensemble!

Now evaluate the ensemble on the test set.

For each image in the test set, make predictions with all your classifiers, then feed the predictions to the blender to get the ensemble's predictions.

How does it compare to the voting classifier you trained earlier?

(Exercise 9 in Chapter 7 of Géron, 2019)

Exercise 2

Use the same datasets from Exercise 1, and do the same exercise using XGBoost and Stacking.

1. What major observations you have? Which one is the best model?
2. Compare feature importance by XGBoost with Random Forests. Describe and discuss your results.