# Practical machine learning

## BOOSTING ALGORITHMS

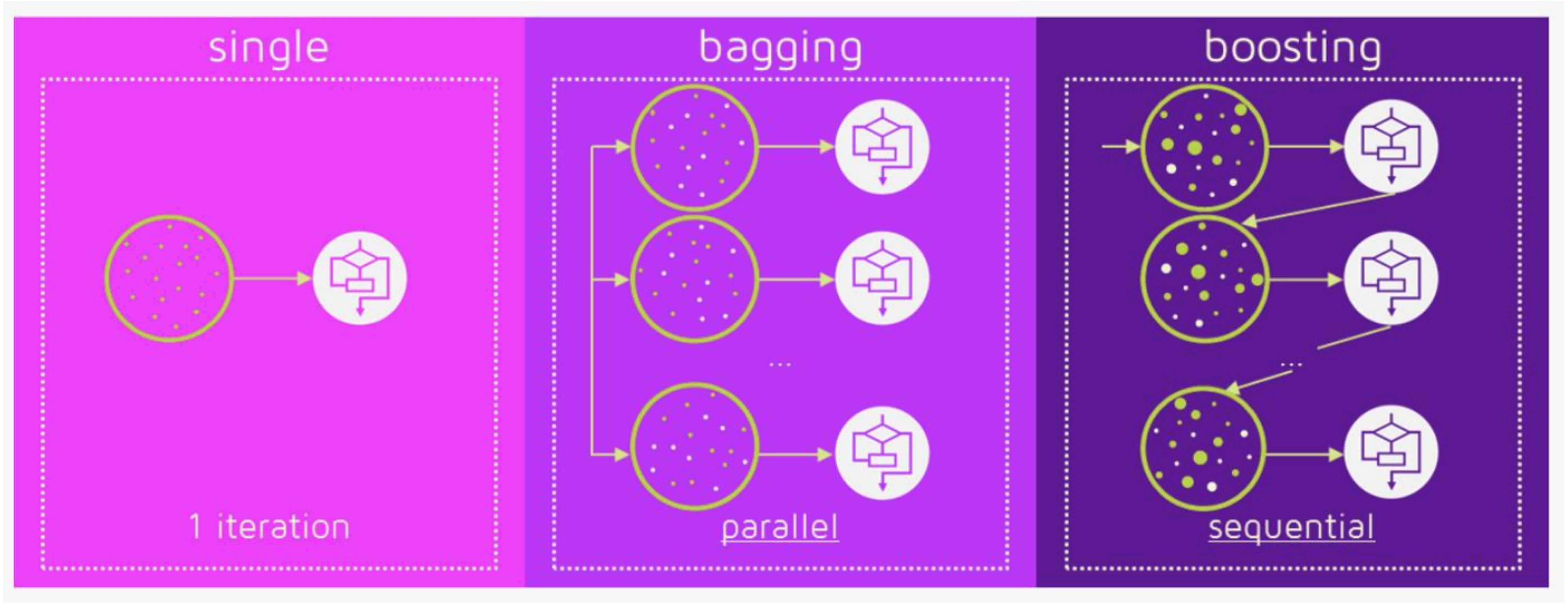
## BOOSTING ALGORITHMS

## ENSEMBLE PREDICTIONS - RECAP

The three most popular methods for combining the predictions from different models are:

- **Bagging.** Building multiple models (typically of the same type) from different subsamples of the training dataset.
  - Random Forest
- **Boosting.** Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.
  - Gradient Boosting
- **Stacking.** Building multiple models (typically of differing types) **and** supervisor model that learns how to best combine the predictions of the primary models.
  - C5.0
  - Logistic Regression
  - k-Nearest Neighbors (kNN)
  - Support Vector Machines (SVM)

# ENSEMBLE PREDICTIONS…



Ref: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

## BOOSTING ALGORITHMS

- Combines multiple weak predictors to build a strong predictor iteratively.
- Boosted algorithms are used where we have plenty of data to make a prediction.
- We seek exceptionally high predictive power.
- It is used for reducing bias and variance.
- **Algorithms:**
    1. Adaptive Boosting (AdaBoost)
    2. **Gradient Boosting Machine (GBM)**
    3. **Extreme Gradient Boosting (XGBoost)**

## 1. AdaBoost

## ADABOOST

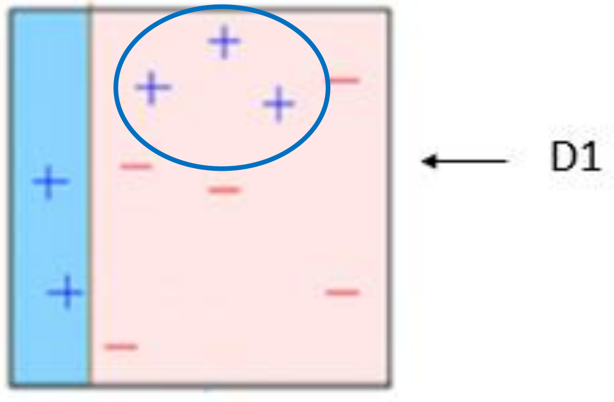
- First really successful boosting algorithm developed.
- It is generally used for classification rather than regression.
- Mostly used with decision tree/decision stumps (A DT with only one level, i.e. only one rule to make a decision.)

## ADABOOST – EXAMPLE

**Example: Classify a data of + and -**

**Week Learner1:**

You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or – (minus).



We see that, this vertical line has incorrectly predicted three + (plus) as – (minus). In such case, we'll **assign higher weights to these three + (plus) and apply another decision stump.**

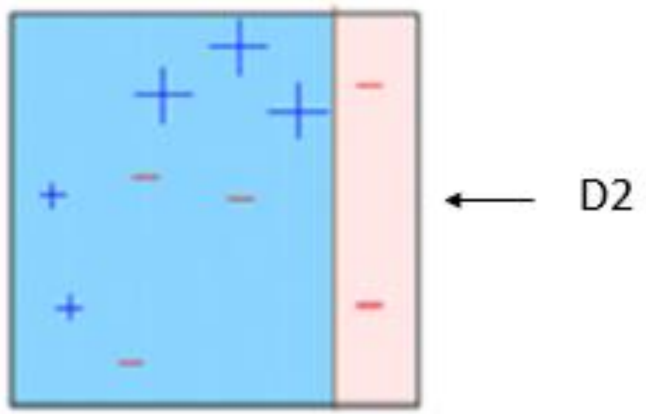
## ADABOOST – PROBLEMS/REQUIREMENTS

**Example:** **Classify a data of +s and -s**

**Week Learner2:**

**the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points.** In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly.

But again, it has caused mis-classification errors. This time with three -(minus). Again, we will assign higher weight to three – (minus) and apply another decision stump.



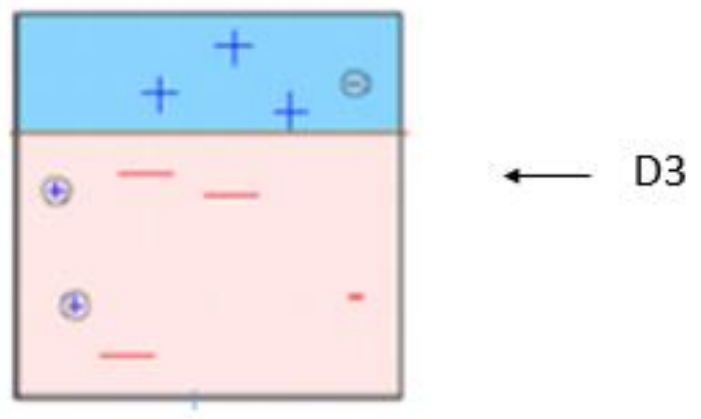
## ADABOOST – EXAMPLE

**Example: Classify a data of +s and -s**

**Week Learner3:**

Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly.

This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of mis-classified observation.

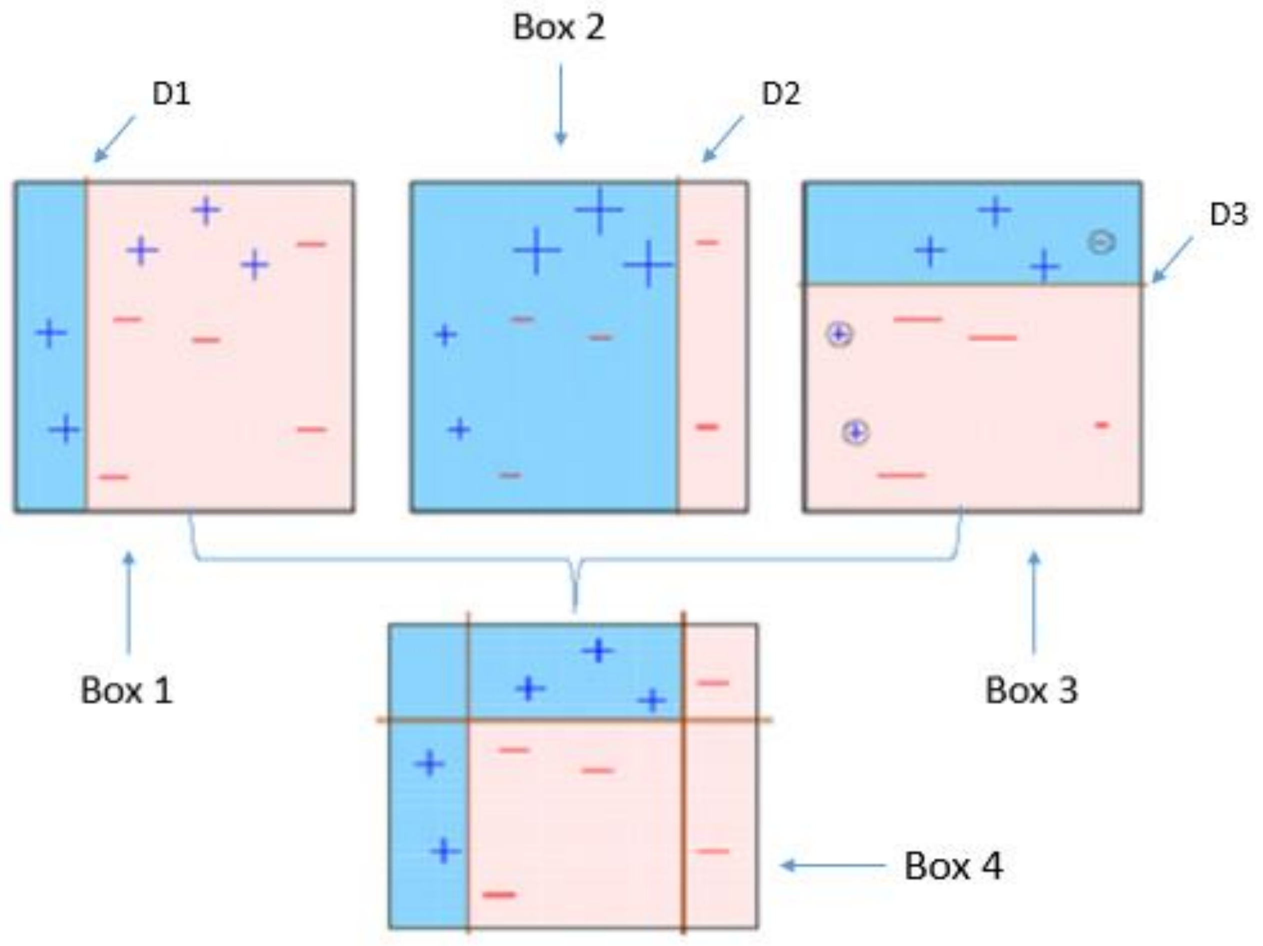


## ADABOOST – PROBLEMS/REQUIREMENTS

**Example: Classify a data of +s and -s**

**Combining 3 week learners:**

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 – HOW TO BUILD A MODEL?

**1. A weak classifier (Ex: decision stump) is prepared on the training data using the weighted samples.**

- Each decision stump makes one decision on one input & outputs a +1.0 or -1.0.

**2. The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:**

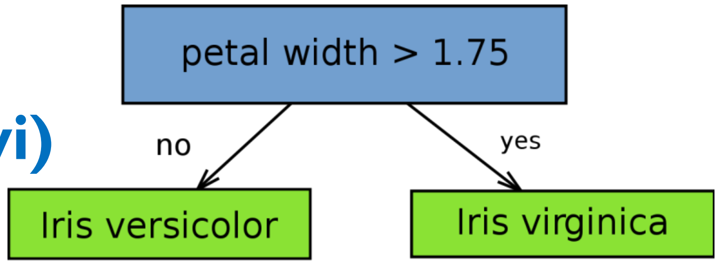
**Error (E) = (N – C) / N**

N – Total samples, C – Correct classifications.

Example: 70 out of 100 are classified correctly by a DT, then Error = (100 – 70)/100 = 0.3

3. **We can modify this by introducing weights**

**Misclassification Rate (MR) = sum(wi * Terror(i)) / sum(wi)**



w(i) – Weight of ith sample,

Terror(i) - is the prediction error for training sample i which is 1 if misclassified and 0 if correctly classified, sum(w) – Sum of all the weights

## ADABOOST – HOW TO BUILD A MODEL?

- For example, if we had **3 training samples** with the **weights 0.01, 0.5 and 0.2**. The predicted labels were A, B and C, and the actual labels were A, A and C, then the terrors would be 0, 1, and 0 (Because B is misclassified - Error is 1)
- The misclassification rate would be calculated as:

  MR = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2) = 0.704

4. **A stage value (Exponential Error Function) is calculated for the trained model which provides a weight for any predictions that the model makes. The stage value for a trained model is calculated as follows:**

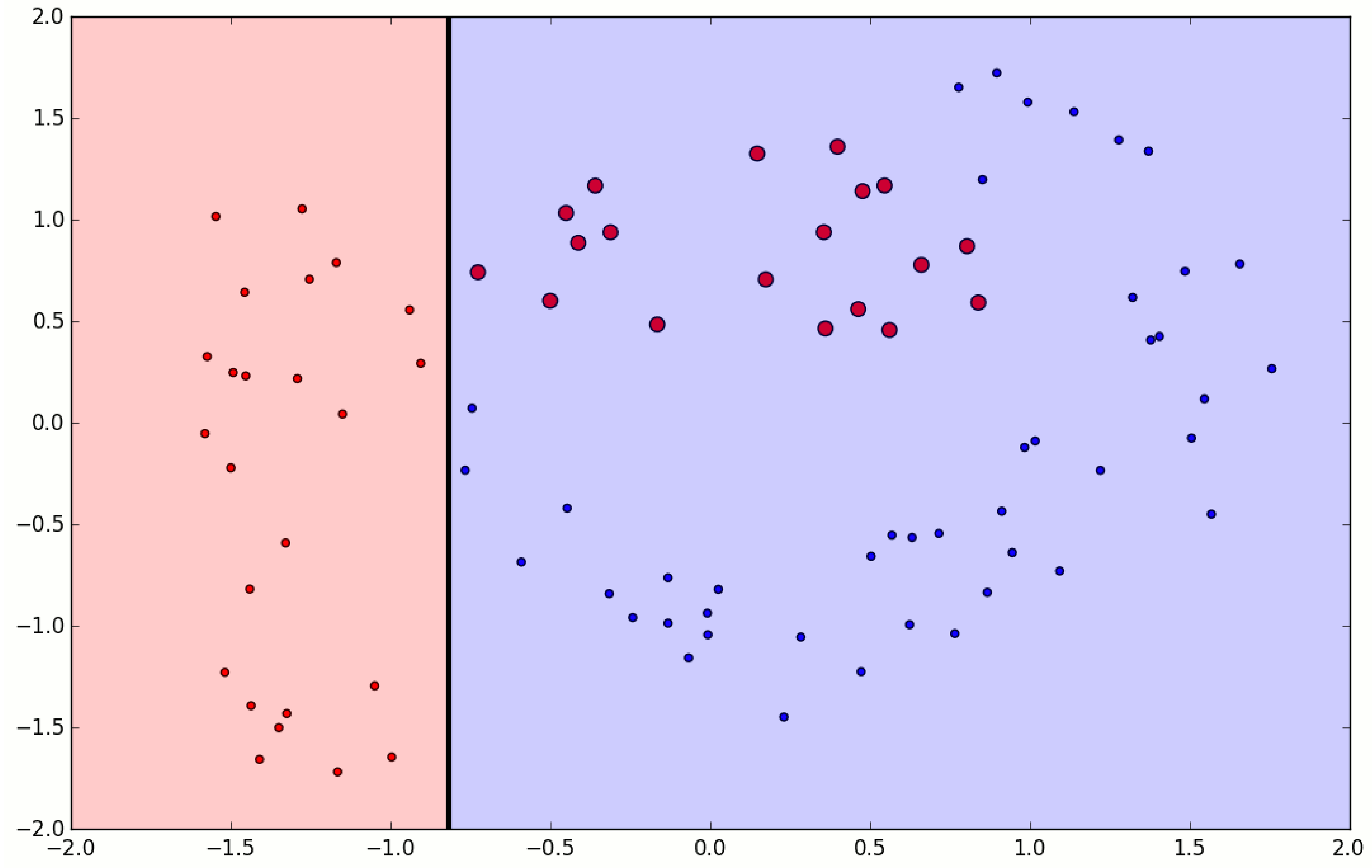
stage = ln((1-MR) / MR)

The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction.

5. **The training weights are updated giving more weight to incorrectly predicted samples, and less weight to correctly predicted samples.**

→ w = w * exp(stage * terror)

# ADABOOST – HOW



## 2. Gradient Boost Machine (GBM)

## GRADIENT BOOSTING

**Gradient Boosting = Gradient Descent + Boosting**

## GRADIENT DESCENT

**Gradient Descent???**

**Gradient – An inclined part of a Road**

**Descent – Moving downward**

## GRADIENT DESCENT…

**Gradient Descent???**

**Gradient** – An inclined part of a Road – **Error (Residual)**

**Descent** – Moving downward - **Minimize**

## Gradient Descent – Minimizing Wrongness/Error

Let h is our predictor function or model.

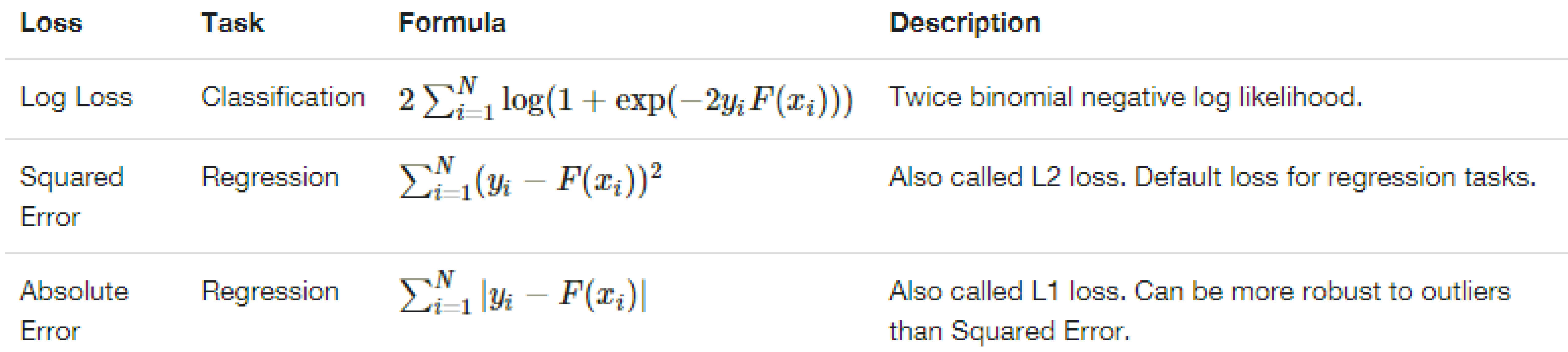
$$h(x_1, x_2, x_3, x_4) = \theta_0 + \theta_1 x_1 + \theta_2 x_3^2 + \theta_3 x_3 x_4 + \theta_4 x_1^3 x_2^2 + \theta_5 x_2 x_3^4 x_4^2$$

How do we make sure $\theta_0$ and $\theta_1$ are getting better with each step ?

Reduce Error/Wrongness - Which is called as Cost Function $J(\theta_0, \theta_1)$

## GRADIENT DESCENT…

### General Cost Functions



| Loss | Task | Formula | Description |
|---|---|---|---|
| Log Loss | Classification | $2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$ | Twice binomial negative log likelihood. |
| Squared Error | Regression | $\sum_{i=1}^{N}(y_i-F(x_i))^2$ | Also called L2 loss. Default loss for regression tasks. |
| Absolute Error | Regression | $\sum_{i=1}^{N}\lvert y_i-F(x_i)\rvert$ | Also called L1 loss. Can be more robust to outliers than Squared Error. |

## GRADIENT DESCENT…

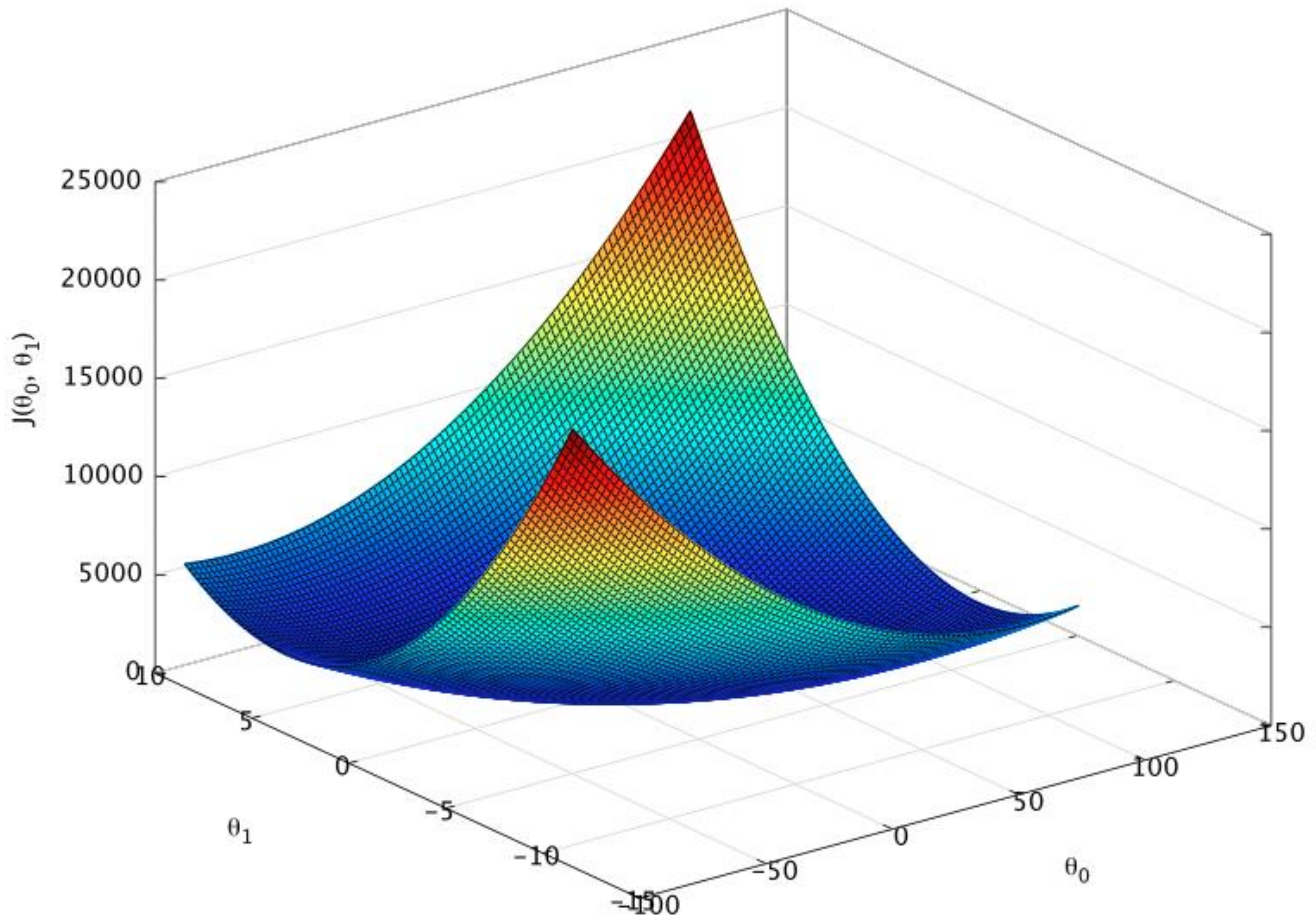
**We use another Cost/Loss Function – Linear Least Squares**

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_{t,i}) - y)^2$$

- The penalty for a bad guess goes up quadratically with the difference between the guess and the correct answer
- So it acts as a very "strict" measurement of wrongness.
- The cost function computes an **average penalty** over all of the training examples.

## GRADIENT DESCENT…

**Assume that the following is the plot of our cost function**

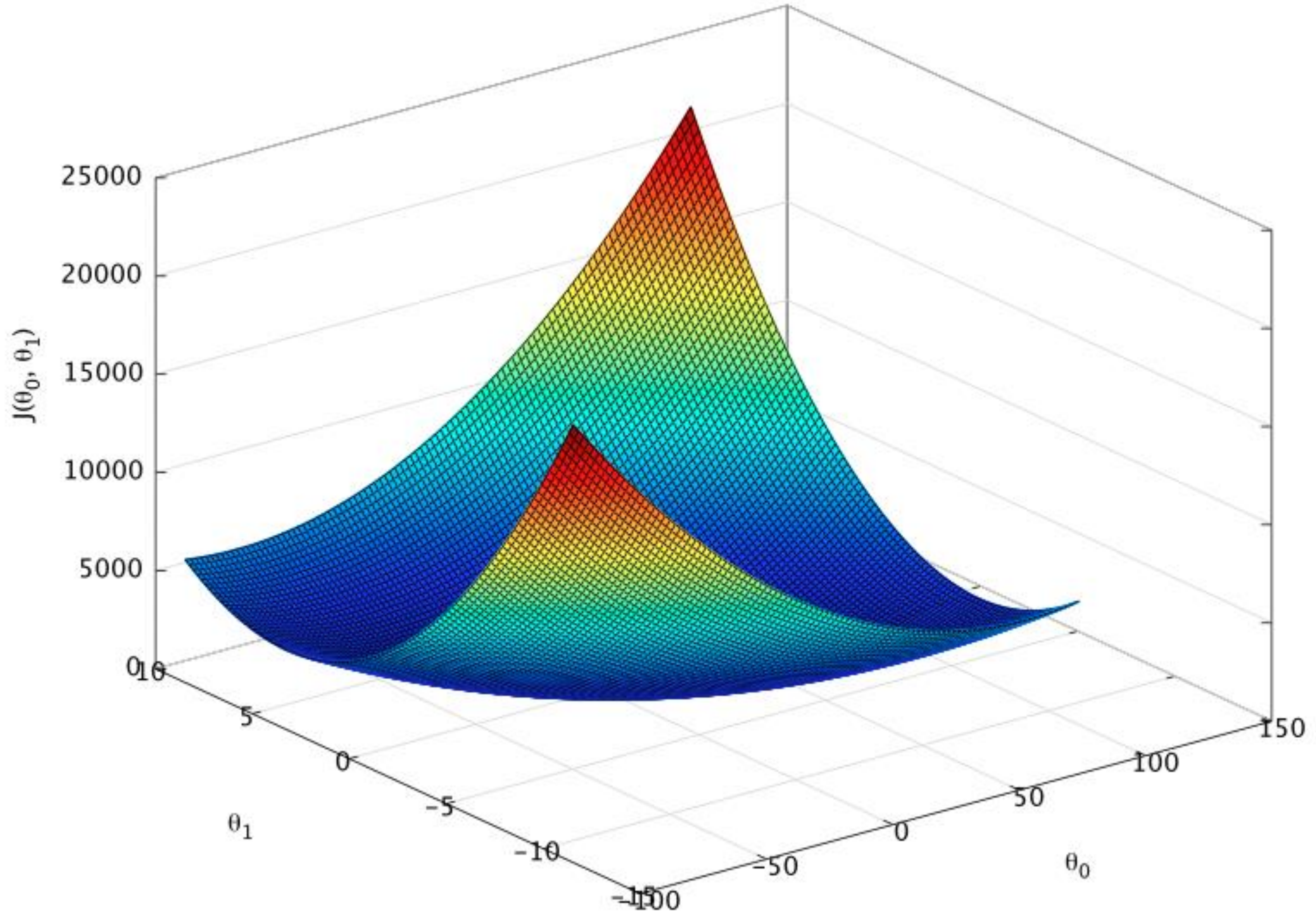


## GRADIENT DESCENT…

The bottom of the bowl represents the lowest cost of our predictor.

The goal is to "roll down the hill", and find the point where the cost function is least.

It requires the calculation of gradient using derivatives.



## GRADIENT BOOSTING

**Gradient Boosting = Gradient Descent + Boosting**

- Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

- boosting can be interpreted as an optimization algorithm on a suitable cost function.

- It is iterative, i.e., algorithms that optimize a cost *function* iteratively, which points in the negative gradient direction.

- Combines weak learners into a single strong learner, in an iterative fashion.

## GRADIENT BOOSTING – WHERE TO USE?

A powerful machine learning algorithm, it can do:

- Regression
- Classification

## AN EXAMPLE

**Predict the age of a person based on:**

whether he likes

- Gardening,
- Playing video games
- Hats

# AN EXAMPLE…

## Data

| PersonID | Age | LikesGardening | PlaysVideoGames | LikesHats |
|---|---|---|---|---|
| 1 | 13 | FALSE | TRUE | TRUE |
| 2 | 14 | FALSE | TRUE | FALSE |
| 3 | 15 | FALSE | TRUE | FALSE |
| 4 | 25 | TRUE | TRUE | TRUE |
| 5 | 35 | FALSE | TRUE | TRUE |
| 6 | 49 | TRUE | FALSE | FALSE |
| 7 | 68 | TRUE | TRUE | TRUE |
| 8 | 71 | TRUE | FALSE | FALSE |
| 9 | 73 | TRUE | FALSE | TRUE |

**Intuitively, we might expect:**

– The people who like gardening are probably older

– The people who like video games are probably younger

– *LikesHats* is probably just random noise (may not be useful)

## AN EXAMPLE…

**Inspect the data…**

| **Feature** | **FALSE(AGE)** | **TRUE(AGE)** |
|---|---|---|
| LikesGardening | {13, 14, 15, 35} | {25, 49, 68, 71, 73} |
| PlaysVideoGames | {49, 71, 73} | {13, 14, 15, 25, 35, 68} |
| LikesHats | {14, 15, 49, 71} | {13, 25, 35, 68, 73} |

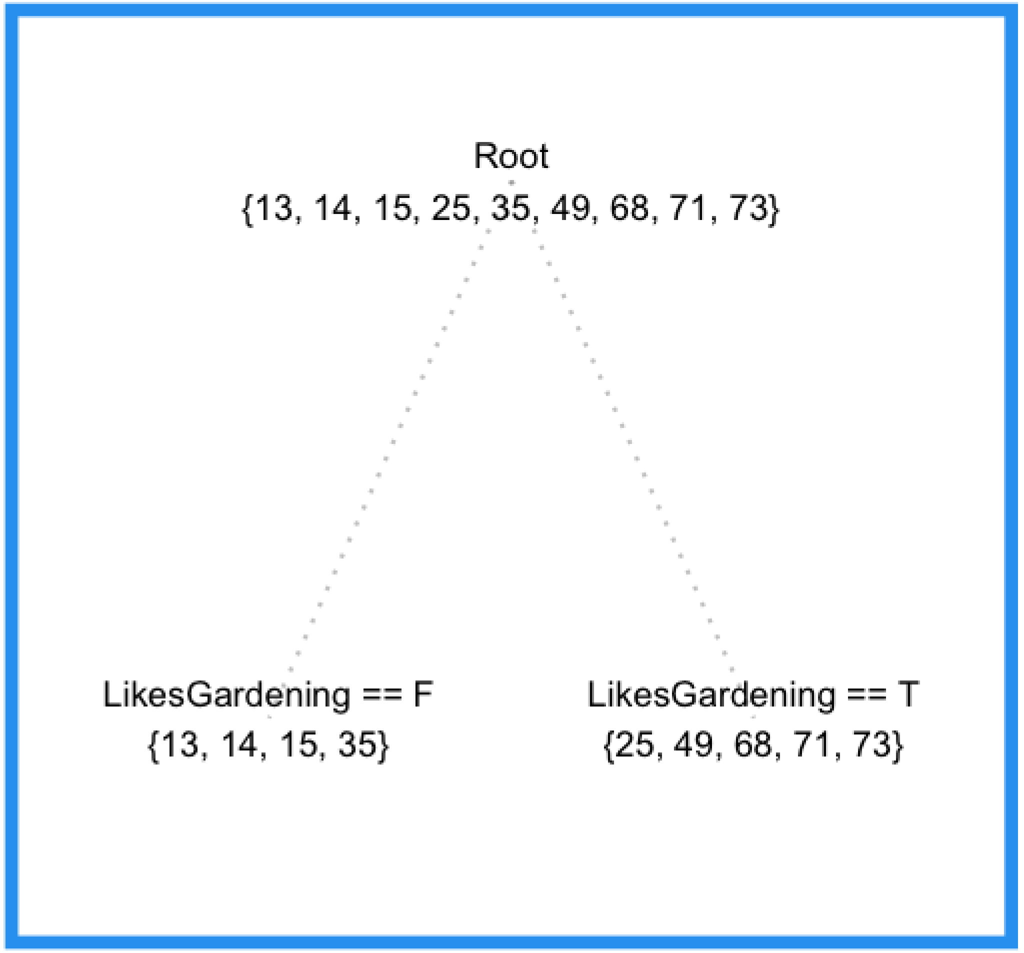
**Intuitively, we might expect:**
– The people who like gardening are probably older → {25, 49, 68, 71, 73}
– The people who like video games are probably younger → {13, 14, 15, 25, 35, 68}
– *LikesHats* is probably just random noise (may not be useful) – It has both young and old people in both of the classes.

**My assumptions are not true…..**

## AN EXAMPLE…

**So, model the data using a Decision Tree – DT1**

Tree 1



**Tree1 is purely based on one feature "Likes Gardening", it is missing very important feature "Likes video games" (so, it is a week learner)**

## AN EXAMPLE…

**measure the training errors from our first tree – Tree1**

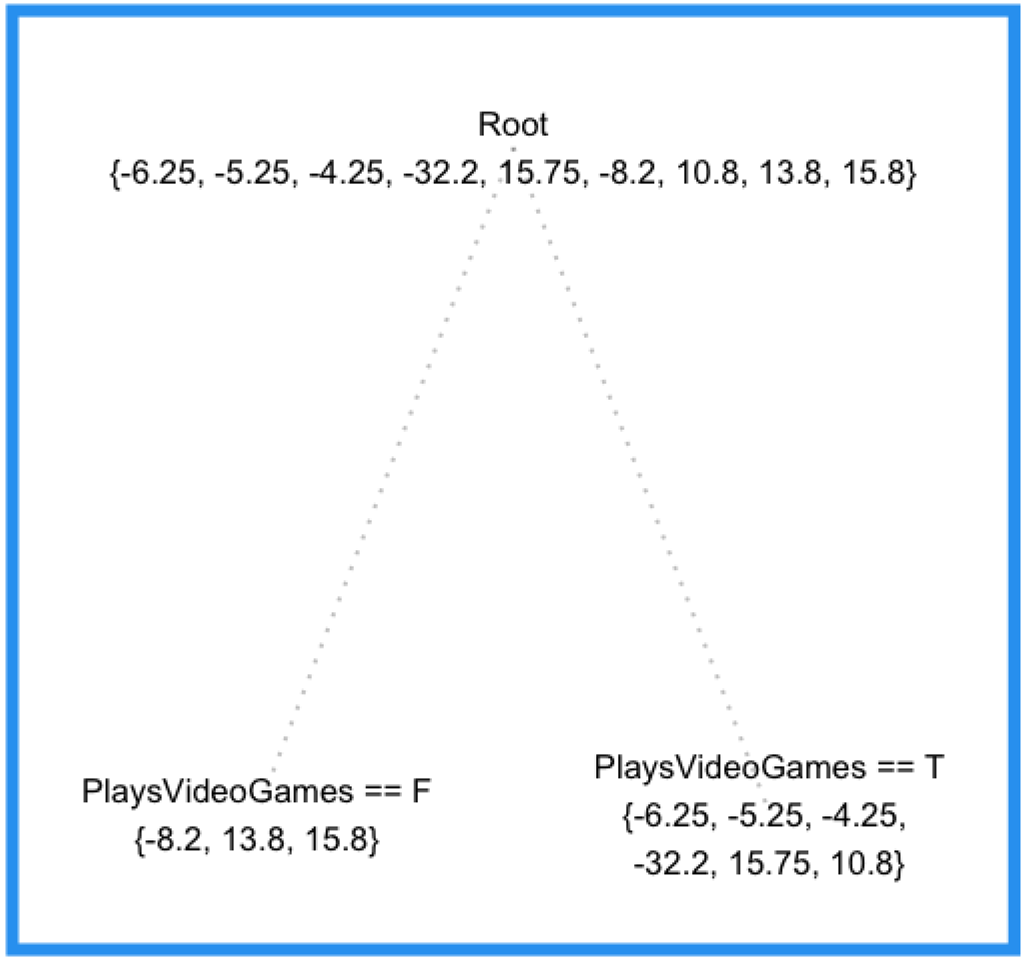
| PersonID | Age | Tree1 Prediction | Tree1 Residual |
|---|---|---|---|
| 1 | 13 | 19.25 | -6.25 |
| 2 | 14 | 19.25 | -5.25 |
| 3 | 15 | 19.25 | -4.25 |
| 4 | 25 | 57.2 | -32.2 |
| 5 | 35 | 19.25 | 15.75 |
| 6 | 49 | 57.2 | -8.2 |
| 7 | 68 | 57.2 | 10.8 |
| 8 | 71 | 57.2 | 13.8 |
| 9 | 73 | 57.2 | 15.8 |

Let us **assume** that the predictions of Tree1 are 19.25, 19.25, 19.25, 57.2 and so on.

## AN EXAMPLE…

Now we can fit a second regression tree to the residuals of the first tree.

Tree2



**Tree2 is purely based on one feature "Plays video Games", it is missing very important feature "Likes Gardening" (so, it is a week learner)**

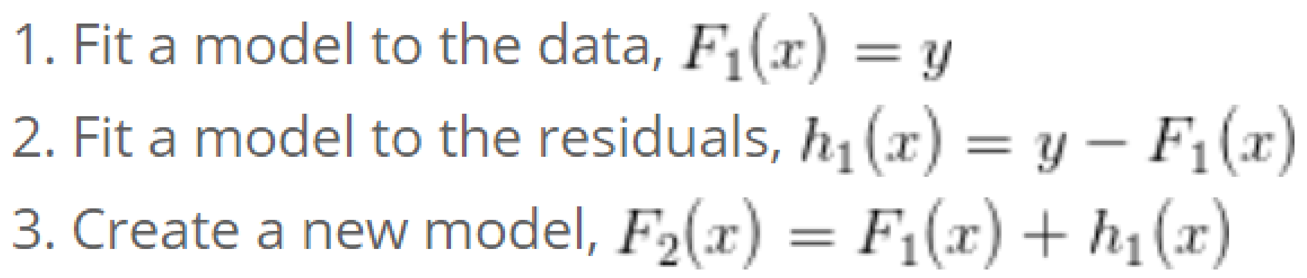
## AN EXAMPLE…

Now we can improve the predictions from our first tree by adding the "error-correcting" predictions from this tree.

| PersonID | Age (A) | Tree1 Prediction (P1) | Tree1 Residual R1 = (A-P1) | Tree2 Prediction (R2) | Combined Prediction (P1+R2) | Final Residual (R2-R1) |
|---|---|---|---|---|---|---|
| 1 | 13 | 19.25 | -6.25 | -3.567 | 15.68 | 2.683 |
| 2 | 14 | 19.25 | -5.25 | -3.567 | 15.68 | 1.683 |
| 3 | 15 | 19.25 | -4.25 | -3.567 | 15.68 | 0.6833 |
| 4 | 25 | 57.2 | -32.2 | -3.567 | 53.63 | 28.63 |
| 5 | 35 | 19.25 | 15.75 | -3.567 | 15.68 | -19.32 |
| 6 | 49 | 57.2 | -8.2 | 7.133 | 64.33 | 15.33 |
| 7 | 68 | 57.2 | 10.8 | -3.567 | 53.63 | -14.37 |
| 8 | 71 | 57.2 | 13.8 | 7.133 | 64.33 | -6.667 |
| 9 | 73 | 57.2 | 15.8 | 7.133 | 64.33 | -8.667 |

## GRADIENT BOOST…

## Coming back to math….



1. Fit a model to the data, $F_1(x) = y$
2. Fit a model to the residuals, $h_1(x) = y - F_1(x)$
3. Create a new model, $F_2(x) = F_1(x) + h_1(x)$

## GRADIENT BOOSTING (FOR REGRESSION) – HOW IT WORKS?

- You are given (x1, y1), (x2, y2), …. (xn, yn), and the task is to fit a model F(x) to minimize a cost function (square loss).
- Let the initial model  is F.
- You check this model and the model is good but not perfect.
- There are some Errors:

  F(x1) = 0.8, while y1 = 0.9, and

  F(x2) = 1.4 while y2 = 1.3

  …

  **How can you improve this model?**

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

**Rules**

- You are not allowed to remove anything from F or change any parameter in F.
- You can add an additional model (regression tree) h to F, so the new prediction will be:

$$F(x) + h(x)$$

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

**Simple solution**

You wish to improve the model such that

$$F(x1) + h(x1) = y1$$

$$F(x2) + h(x2) = y2$$

$$\vdots$$

$$F(xn) + h(xn) = yn$$

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

**Simple solution…**

We can write in different form,

$$h(x1) = y1 - F(x1)$$

$$h(x2) = y2 - F(x2)$$

$$\vdots$$

$$h(xn) = yn - F(xn)$$

Can any regression tree "h" achieve this goal perfectly?

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

**Simple solution…**

We can write in different form,

h(x1) = y1 - F(x1)

h(x2) = y2 - F(x2)

:::

h(xn) = yn - F(xn)

Can any regression tree h achieve this goal perfectly?

Maybe not....But some regression tree might be able to do this approximately.

**How?**

Just fit a regression tree "h" to data, We get a much better model.

$$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \ldots, (x_n, y_n - F(x_n))$$

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

yi - F(xi ) are called residuals.

- These are the parts that existing model F cannot do well.
- The role of "h" is to compensate the shortcoming of existing model F.
- If the new model "F + h" is still not satisfactory, we can add another regression tree ...

We are improving the predictions of training data, is the procedure also useful for test data?

- Yes! Because we are building a model, and the model can be applied to test data as well.

How is this related to gradient descent?

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

### Gradient Descent

- Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

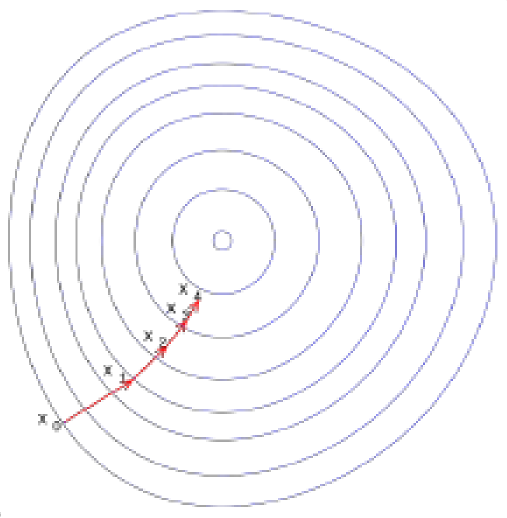


Figure Source: Gradient Descent. Source: http://en.wikipedia.org/wiki/Gradient_descent

## GRADIENT BOOSTING FOR REGRESSION – HOW IT WORKS?

How is this related to gradient descent?

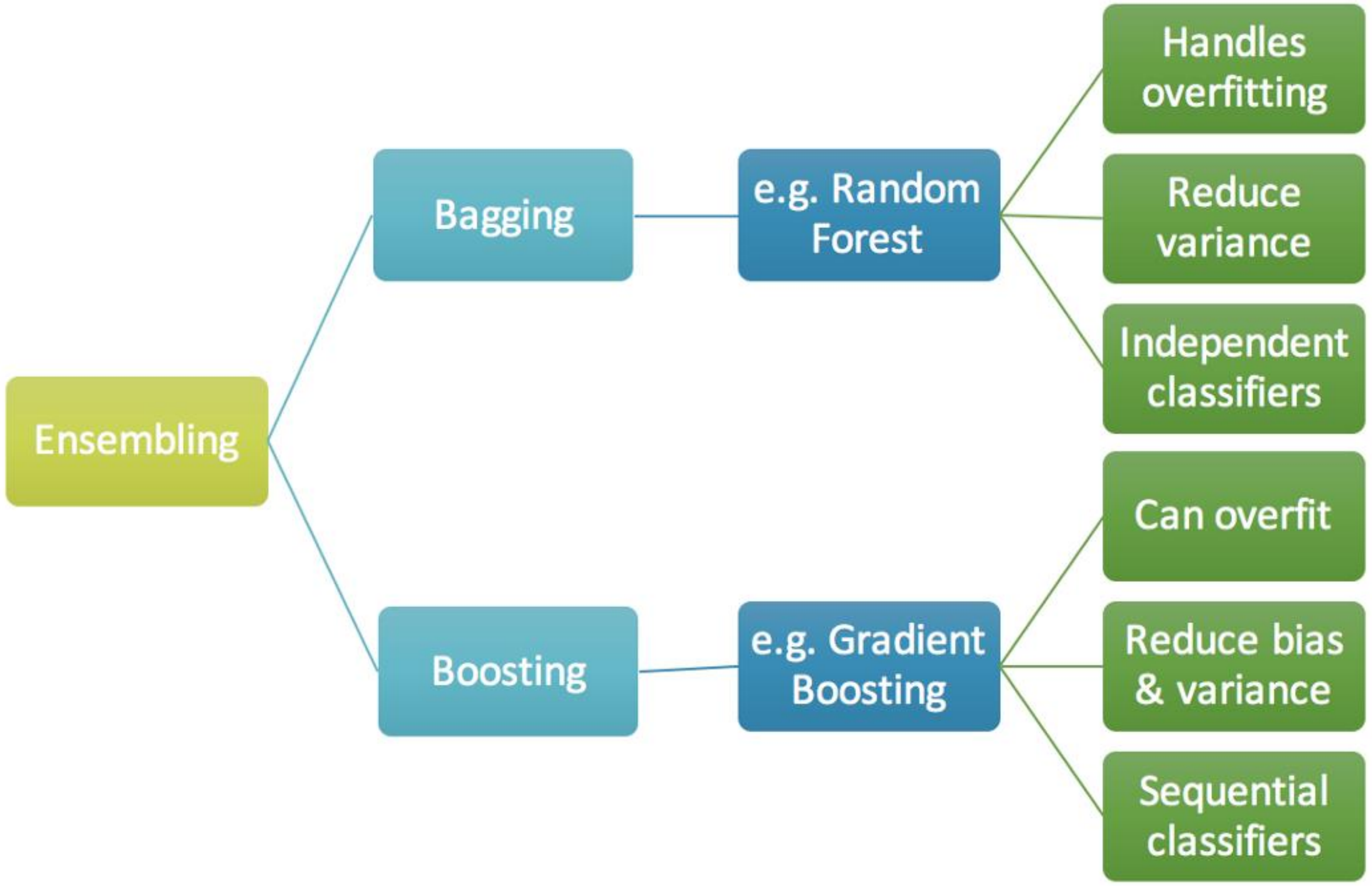
$$F(x_i) := F(x_i) + h(x_i)$$

$$F(x_i) := F(x_i) + y_i - F(x_i)$$

$$F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$

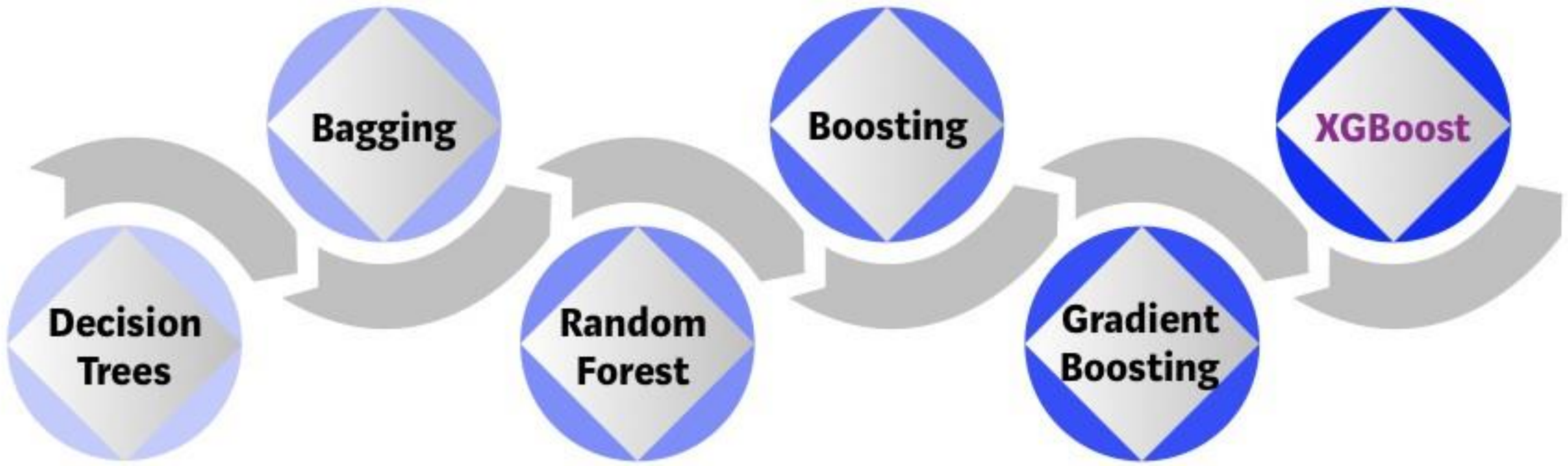
$$\theta_i := \theta_i - \rho\frac{\partial J}{\partial \theta_i}$$

## ENSEMBLE PREDICTIONS - BAGGING .VS. BOOSTING



## XGBOOST

# DECISION TREE TO XGBOOST - EVOLUTION



Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias

A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

## DEFINITION OF BOOSTING - REVISITING

- 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
- For example: Above, we have defined 5 weak learners. Out of these 5, 3 are voted as 'SPAM' and 2 are voted as 'Not a SPAM'. In this case, by default, we'll consider an email as SPAM because we have higher(3) vote for 'SPAM'.

## HOW BOOSTING ALGORITHMS WORKS?

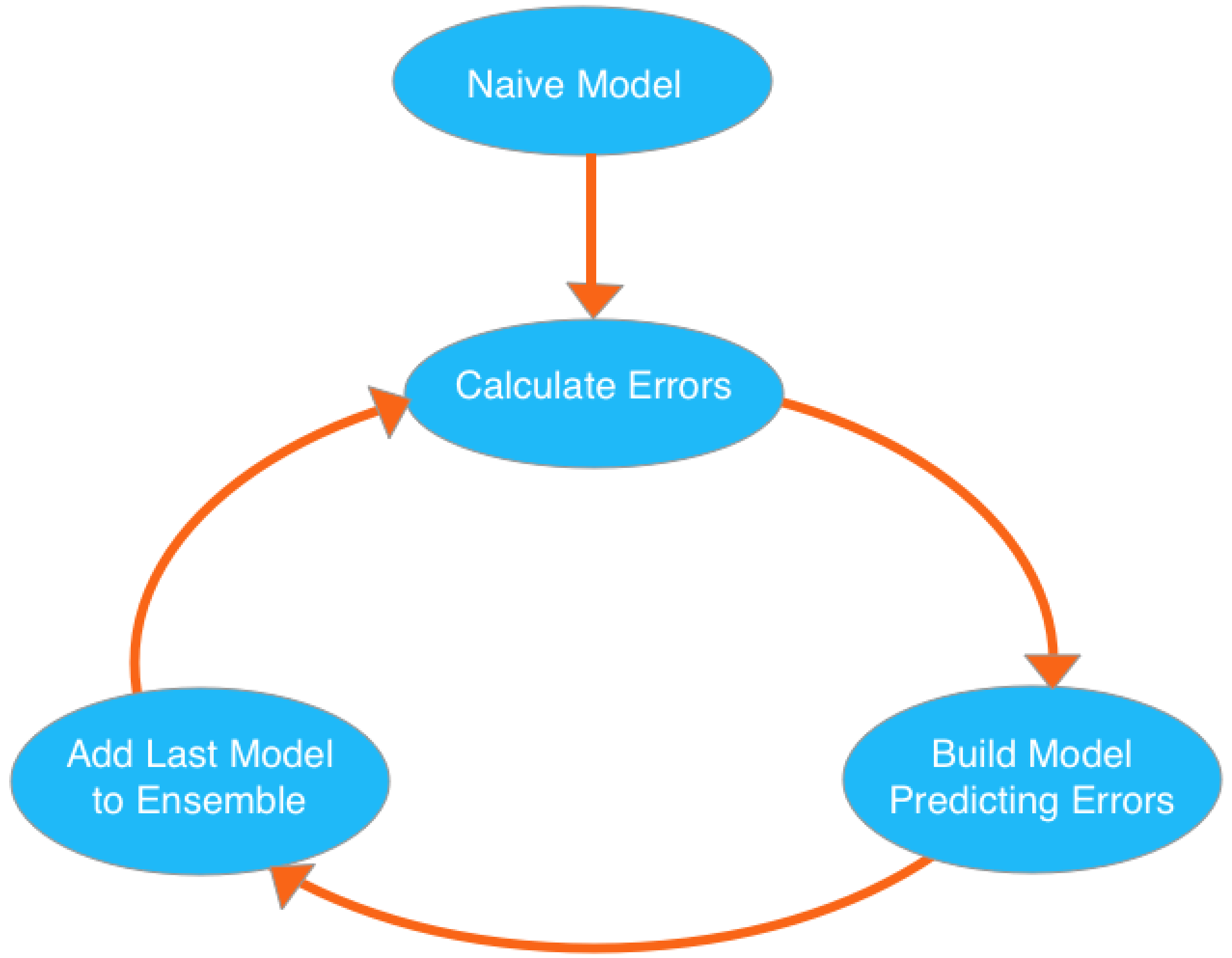
- To find weak rule, we apply base learning algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.
- For choosing the right distribution, here are the following steps:

  *Step 1:* The base learner takes all the distributions and assign equal weight or attention to each observation.

  *Step 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.

  *Step 3:* Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.
- Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are misclassified or have higher errors by preceding weak rules.

## REVISITING GRADIENT BOOSTING - COMPONENTS

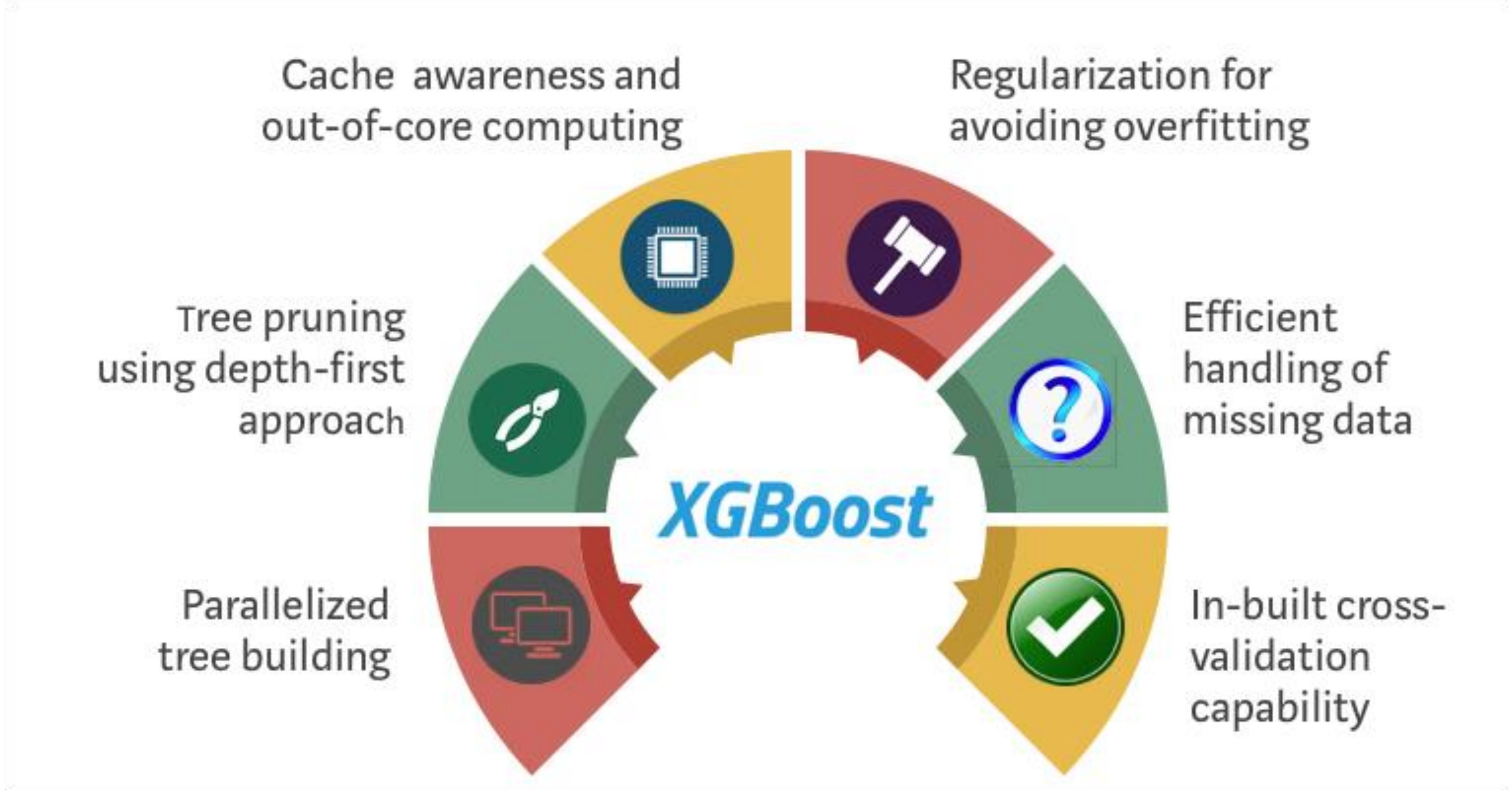


## XGBOOST - INTRODUCTION

- XGBoost stands for eXtreme Gradient Boosting.
- XGBoost is an extension to gradient boosted decision trees (GBM)
- Specially designed to improve speed and performance.

## XGBOOST – FEATURES – WHY IT PERFORMS SO WELL?

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

## XGBOOST – FEATURES – WHY IT PERFORMS SO WELL?

- **Parallelization of tree construction using all of your CPU cores during training.** Collecting statistics for each column can be parallelized, giving us a parallel algorithm for split finding.
- **Cache-aware Access:** XGBoost has been designed to make optimal use of hardware. This is done by allocating internal buffers in each thread, where the gradient statistics can be stored.
- **Blocks for Out-of-core Computation** for very large datasets that don’t fit into memory. optimized to efficiently fetch and access data stored in slow bulk memory such as hard drives or tape drives.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Regularization** for avoiding overfitting, where GBM has no provision.
- Can handle missing values

## CATBOOST

## XGBOOST – FEATURES – WHY IT PERFORMS SO WELL?

- CatBoost is an algorithm for **gradient boosting on decision trees.**
- It is developed by Yandex researchers and engineers.
- CatBoost can be used for solving problems, such as:
  - classification (binary, multi-class)
  - regression
  - Ranking
- While deep learning algorithms require lots of data and computational power, boosting algorithms are still needed for most business problems.
- However, boosting algorithms like XGBoost takes hours to train, and sometimes you’ll get frustrated while tuning hyper-parameters.

## MAIN ADVANTAGES

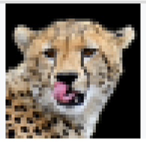
1 **Great quality without parameter tuning**

Reduce time spent on parameter tuning, because CatBoost provides great results with default parameters

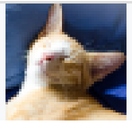
2

**Categorical features support**

Improve your training results with CatBoost that allows you to use non-numeric factors, instead of having to pre-process your data or spend time and effort turning it to numbers.

3  **Fast and scalable GPU version**

Train your model on a fast implementation of gradient-boosting algorithm for GPU. Use a multi-card configuration for large datasets.

4  **Improved accuracy**

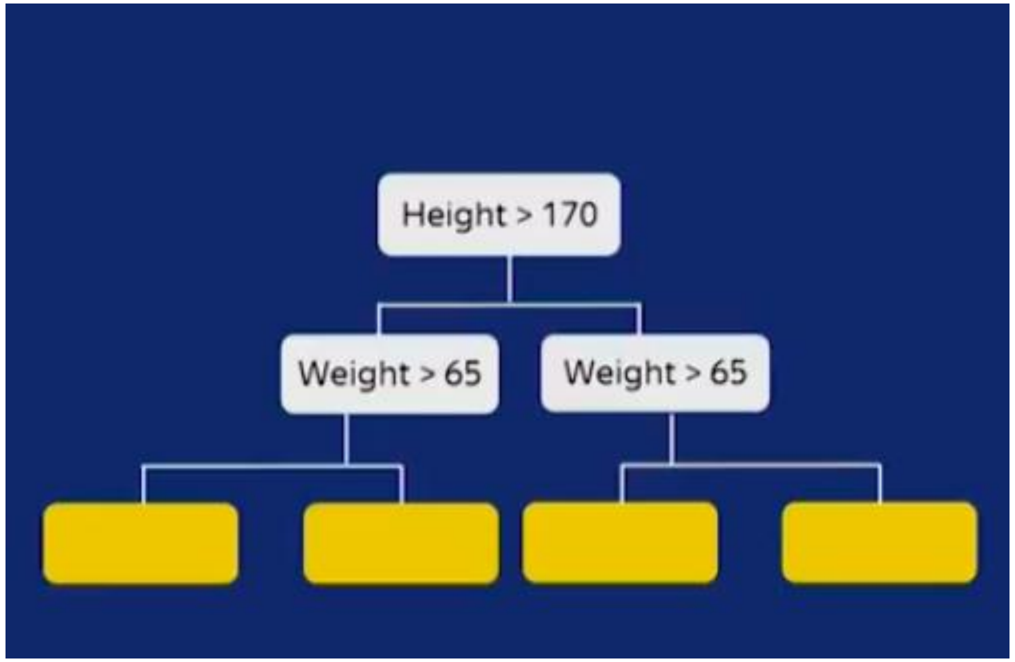
Reduce overfitting when constructing your models with a novel gradient-boosting scheme.

5 **Fast prediction**

Apply your trained model quickly and efficiently even to latency-critical tasks using CatBoost's model applier

## CATBOOST – TREE STRUCTURE

- One main difference between CatBoost and other boosting algorithms is that the **CatBoost implements symmetric trees**.
- It helps in decreasing prediction time, which is extremely important for low latency environments.



**Example of a symmetric tree**

## PROCEDURE FOR OTHER BOOSTING METHODS (EX: GBM)

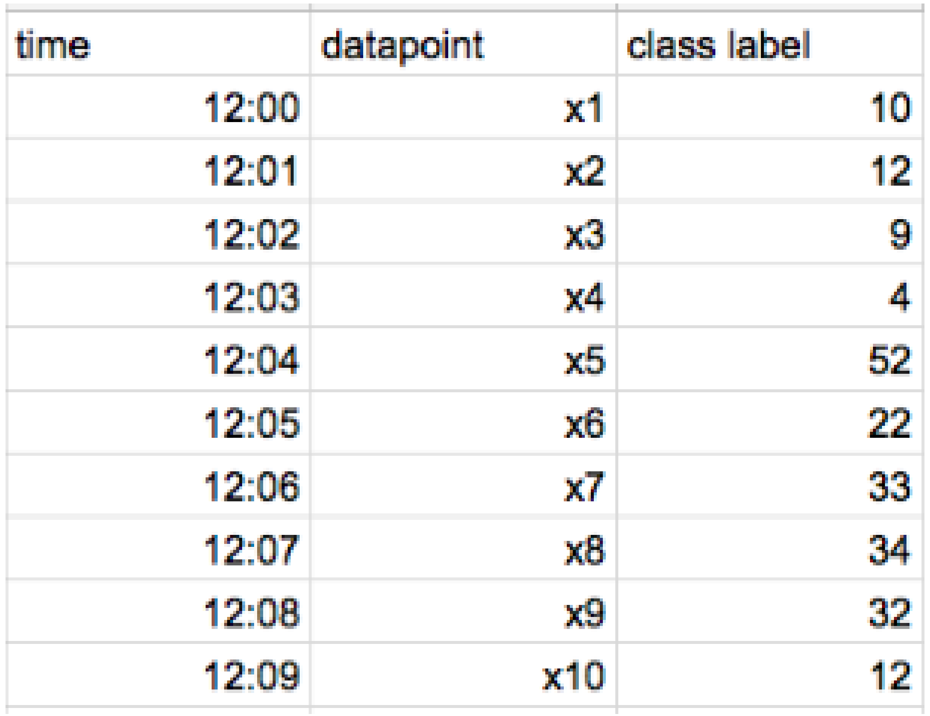
- Step 1: Consider all (or a sample ) the data points to train a highly biased model.
- Step 2: Calculate residuals (errors) for each data point.
- Step 3: Train another model with the *same* data points by considering residuals (errors) as target values.
- Step 4: Repeat Step 2 & 3 ( *for n iterations*).

**Problem with this procedure:**
This procedure is prone to overfitting, because we are calculating residuals of each data point by using the model that has already been trained on same set of data points.

## PROCEDURE OF CATBOOST

- CatBoost does gradient boosting in a very elegant manner. Below is an explanation of CatBoost using a toy example.

Let’s say, we have 10 data points in our dataset and are *ordered in time* as shown below.



| time | datapoint | class label |
|---|---|---|
| 12:00 | x1 | 10 |
| 12:01 | x2 | 12 |
| 12:02 | x3 | 9 |
| 12:03 | x4 | 4 |
| 12:04 | x5 | 52 |
| 12:05 | x6 | 22 |
| 12:06 | x7 | 33 |
| 12:07 | x8 | 34 |
| 12:08 | x9 | 32 |
| 12:09 | x10 | 12 |

## PROCEDURE OF CATBOOST - EXAMPLE

- **Step 1**: Calculate residuals for each data point using a model that has been trained on all the other data points at that time (*For Example*, *to calculate residual for x5 datapoint, we train one model using x1, x2, x3, and x4* ). We train different models to calculate residuals for different data points. In the end, we are calculating residuals for each data point that the model has never seen before.
- **Step 2**: Train a model by using the residuals of each data point as class values.
- **Step 3**: Repeat Step 1 & Step 2 (*for n iterations*)

**What is the problem in the above procedure?**
For the above toy dataset, we should train 9 different models to get residuals for 9 data points. This is **computationally expensive** when we have many data points.
**Solution: Ordered Boosting**

## PROCEDURE OF CATBOOST – ORDERED BOOSTING

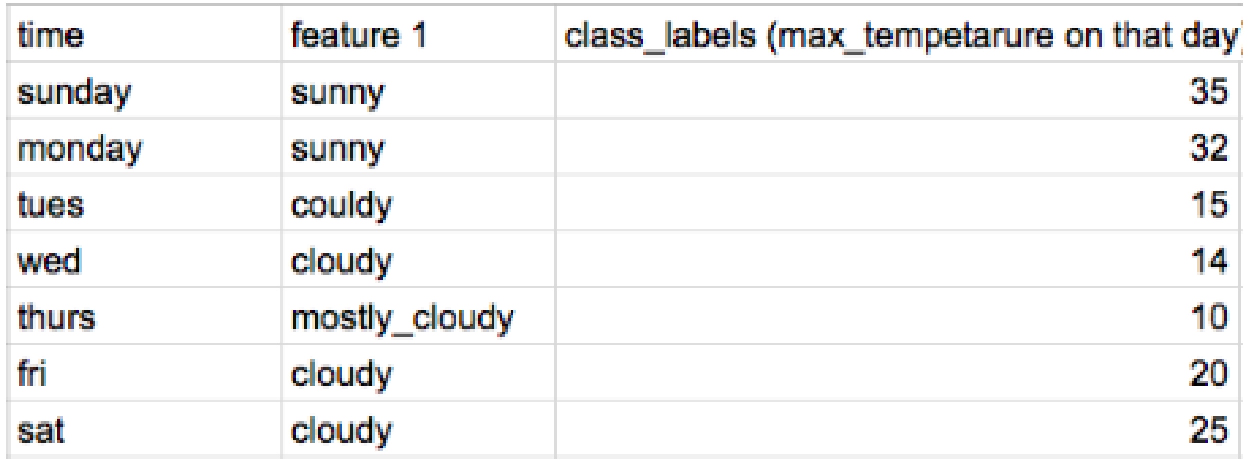
- Instead of training different models for each data point, it **trains only log*(num_of_datapoints)* models**.
- If a model has been trained on *n* data points then that model is used to calculate residuals for the next ***n*** data points.

**Ordered Boosting by CatBoost:**

- A model that has been trained on the first data point is used for calculating residuals of the second data point.
- Another model that has been trained on the first two data points is used for calculating residuals of the third and fourth data points
- and so on…
- *Ex: In the above toy dataset, we calculate residuals of x5,x6,x7, and x8 using a model that has been trained on x1, x2,x3, and x4*.

## CATBOOST – HANDLING CATEGORICAL ATTRIBUTES

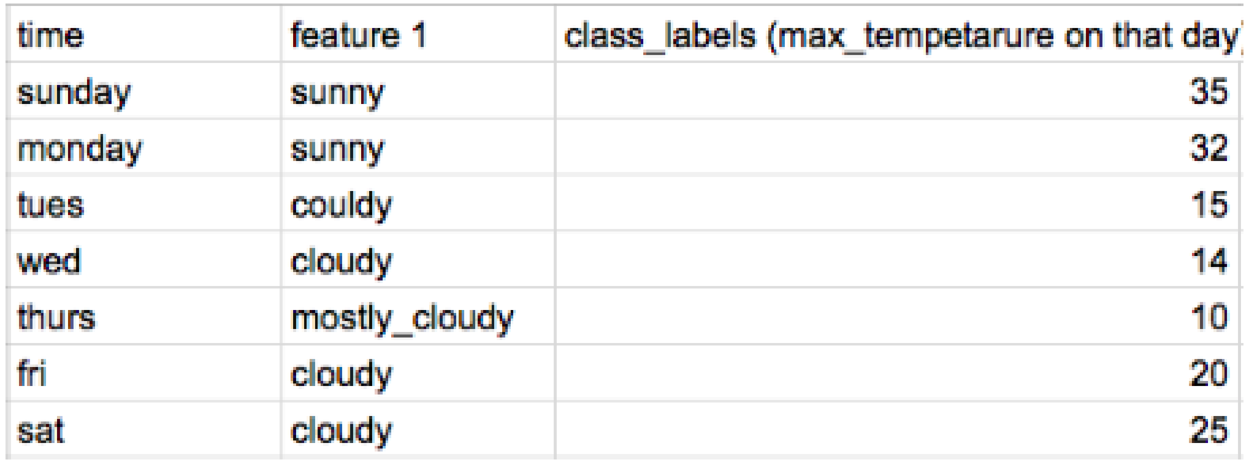
- CatBoost has a very good vector representation of categorical data. It takes concepts of ordered boosting and applies the same to response coding.
- In response coding, we represent categorical features using the mean to the target values of the data points. CatBoost considers only the past data points to calculate the mean value. Below is a detailed explanation with examples.



| time | feature 1 | class_labels (max_tempetarure on that day) |
|---|---|---|
| sunday | sunny | 35 |
| monday | sunny | 32 |
| tues | couldy | 15 |
| wed | cloudy | 14 |
| thurs | mostly_cloudy | 10 |
| fri | cloudy | 20 |
| sat | cloudy | 25 |

## CATBOOST – HANDLING CATEGORICAL ATTRIBUTES…

- CatBoost vectorize all the categorical features without any target leakage. Instead of considering all the data points, it will consider only data points that are past in time. For Example,
- On Friday, it represents cloudy = *(15+14) /2 = 15.5*
- On Saturday, it represents cloudy = *(15+14+20)/3 = 16.3*
- But on Tuesday, it represents cloudy = 0/0?
- It implements **Laplace smoothing** to **overcome zero by zero problem.**



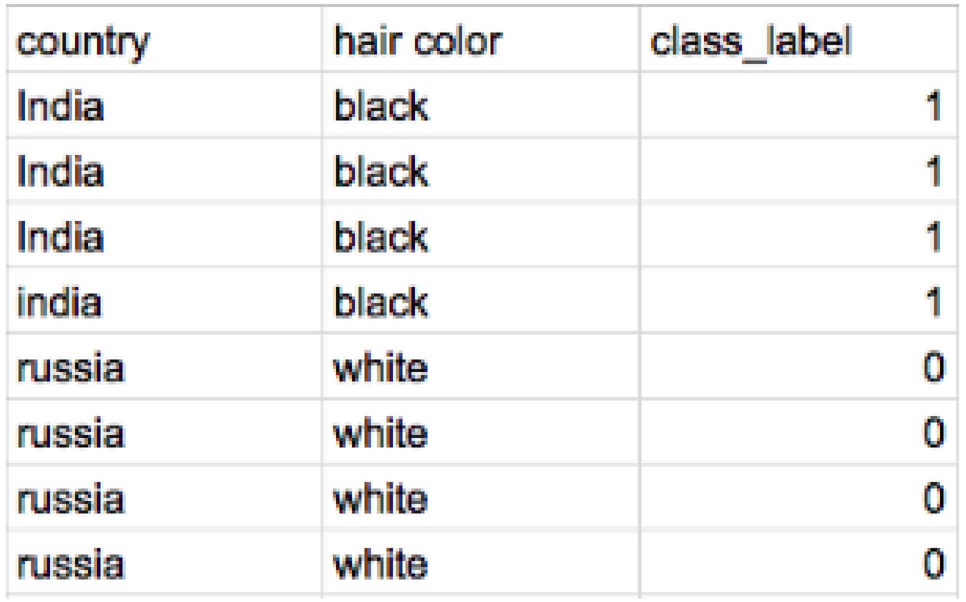
| time | feature 1 | class_labels (max_tempetarure on that day |
|---|---|---|
| sunday | sunny | 35 |
| monday | sunny | 32 |
| tues | couldy | 15 |
| wed | cloudy | 14 |
| thurs | mostly_cloudy | 10 |
| fri | cloudy | 20 |
| sat | cloudy | 25 |

# CATBOOST – CATEGORICAL FEATURE COMBINATIONS

CatBoost combines multiple categorical features.

For the most number of times combining two categorical features makes sense. CatBoost does this for you automatically.

**Example:**
In this dataset, there are two features (country and hair length). We can easily observe that whenever a person is from India, his/her hair color is black. We can represent those two features into a single feature.

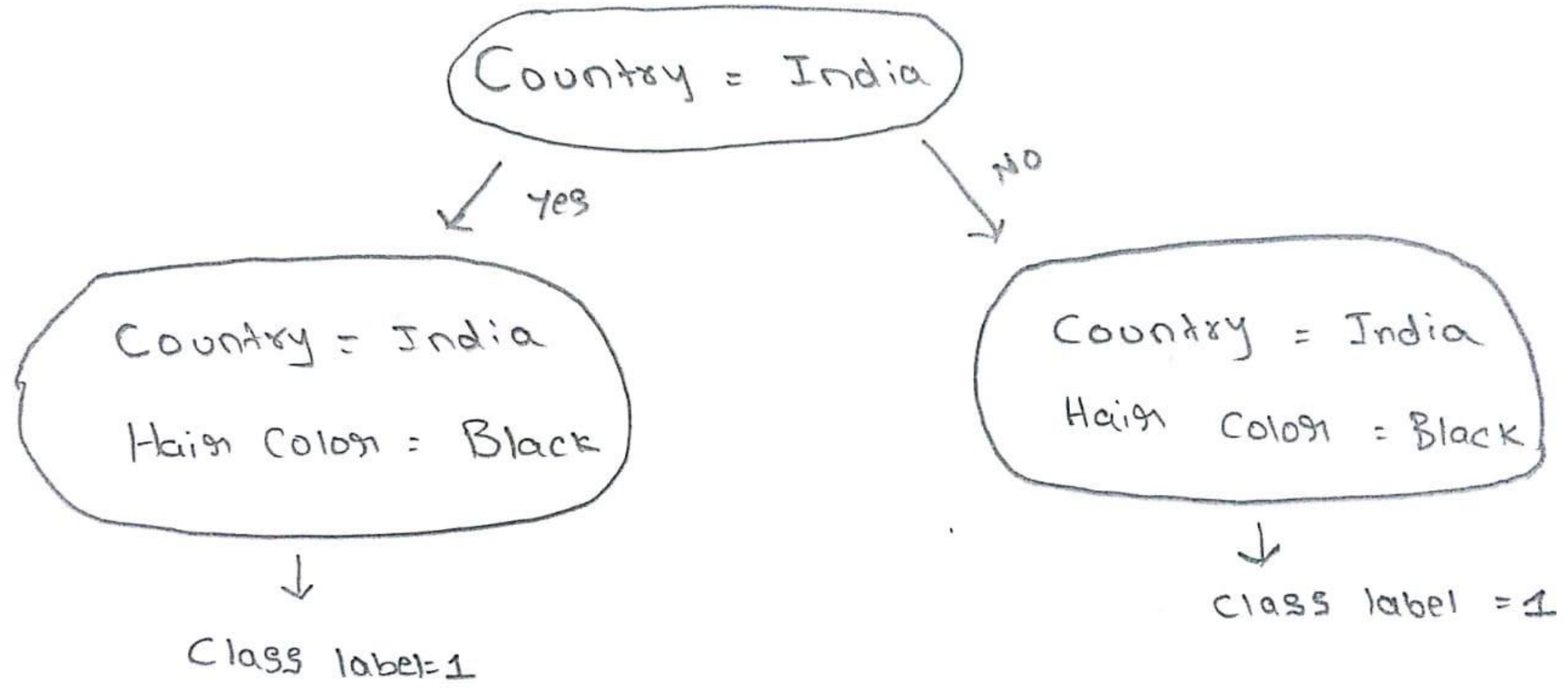


| country | hair color | class_label |
|---|---|---|
| India | black | 1 |
| India | black | 1 |
| India | black | 1 |
| india | black | 1 |
| russia | white | 0 |
| russia | white | 0 |
| russia | white | 0 |
| russia | white | 0 |

## CATBOOST – CATEGORICAL FEATURE COMBINATIONS

CatBoost finds the best possible feature combinations and considers them as a single feature.

Below is the neat diagram of CatBoost representing two features as a single feature at level 2 of the tree.



## CATBOOST – ONE HOT ENCODING

By default, CatBoost represents all the two-categorical features with One-hot encoding.

If you would like to implement One-hot encoding on a categorical feature that has **N** different categories then you can change parameter ***one_hot_max_size*** = ***N****.*

## CATBOOST – HANDLING NUMERICAL FEATURES

CatBoost handles the numerical features in the same way that other tree algorithms do.

We select the best possible split based on the Information Gain.

## CATBOOST – LIMITATIONS

- When the dataset has many numerical features, CatBoost takes so much time to train
- CatBoost *doesn’t* support sparse matrices.

## CATBOOST – OTHER ADVANTAGES

- By default, CatBoost has an overfitting detector that stops training when CV error starts increasing. You can set parameter od_type = Iter to stop training your model after few iterations.
- We can also balance an imbalanced dataset with the class_weight parameter.
- CatBoost not only interprets important features, but it also returns important features for a given data point what are the important features.
- The code for training CatBoost is simply straight forwarded and is similar to the sklearn module. You can go through the documentation of CatBoost here for a better understanding.

## CATBOOST – OTHER ADVANTAGES

**Goodbye to Hyper-parameter tuning?**

CatBoost is implemented by powerful theories like ordered Boosting, Feature Combiinations.

It makes sure that we are not overfitting our model.

It also implements symmetric trees which eliminate parameters like (min_child_leafs ).

We can further tune with parameters like *learning_rate*, *L2_regulariser*, but the results *don't* vary much.

## REFERENCES

https://coderzcolumn.com/tutorials/machine-learning/xgboost-an-in-depth-guide-python

https://github.com/catboost/tutorials/blob/master/python_tutorial.ipynb

# Thank You