XGboost and SHAP values

Data Science Course

Dr. Ariel Mantzura

2025-03-19

Topics to be Covered in this Lecture

- Introduction
- Tree Ensembles
- Regularized Learning Objective
- Additive Training and Gradient Tree Boosting
- SHAP values

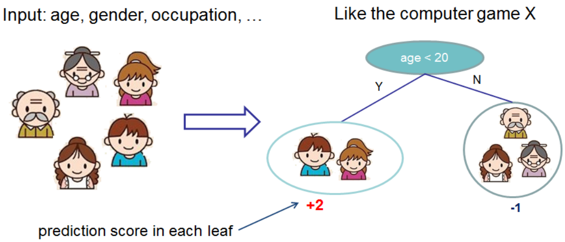
Sources: based upon **XGBoost: A Scalable Tree Boosting System, Tianqi Chen and Carlos Guestrin**

Introduction

- XGBoost stands for “Extreme Gradient Boosting”.
- The term “Gradient Boosting” originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman.
- XGBoost is used for supervised learning problems, where we use the training data x_i (with multiple features) to predict a target variable y_i.
- The prediction value can have different interpretations, depending on the task, i.e., regression or classification.

Introduction - Tree ensembles

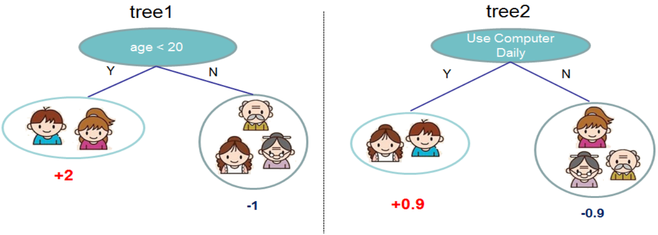
- The model choice of XGBoost is decision tree ensembles.
- The tree ensemble model consists of a set of classification or regression trees (CART).
- Here's a simple example of a CART that classifies whether someone will like a hypothetical computer game X.



Introduction - Tree ensembles

- We classify the members of a family into different leaves, and assign them the score on the corresponding leaf.
- Usually, a single tree is not strong enough to be used in practice.
- What is actually used is the ensemble model, which sums the prediction of multiple trees together.
- Random forests and boosting which we learned are also types of tree ensembles.

Introduction - Tree ensembles



- f() = 2 + 0.9= 2.9 f()= -1 - 0.9= -1.9

Introduction - Tree ensembles

- Here is an example of a tree ensemble of two trees.
- The prediction scores of each individual tree are summed up to get the final score.
- If you look at the example, an important fact is that the two trees try to complement each other

Tree ensembles - Mathematical definition

- Let $\mathcal{D}$ be a data set with n observations and p features and a target variable.

$$\mathcal{D} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} & y_1 \\ x_{21} & x_{22} & \dots & x_{2p} & y_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} & y_p \end{bmatrix}$$

- $x_i \in \mathcal{R}^p$
- x_{ij} - the value of the j^{th} variable for the i^{th} observation

Tree ensembles - Mathematical definition

- A tree ensemble model uses K additive functions to predict the output.
- $$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Tree ensembles - Mathematical definition

- $\mathcal{F}$ is a family of all trees which can be written as follows:

$$\mathcal{F} = \{f(x) = w_{q(x)}\}$$

- $q(x) : R^P \to \{1, , , T\}$
- $w = \{w_1, w_2, ...w_T\} \in R^T$
- $q(x)$ represents the structure of a tree that maps an observation to the corresponding leaf index in $\{1, , , , , T\}$.
- T is the number of leaves in the tree.
- Each f_k corresponds to an independent tree structure q_k and leaf weights w_k.

Regularized Learning Objective

- To learn the set of functions f_k used in the model, we minimize the following regularized objective.
- $$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$
- $l(y_i, \hat{y}_i)$ is a loss function for example the quadratic loss function $(y_i - \hat{y}_i)^2$ so that
  $$\sum_{i=1}^{n} l(y_i, \hat{y}_i) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
  .
- From now on we will use the quadratic loss function.
- $\Omega(f_k)$ is a penalty added on each of the trees.
- Where
  $$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Additive Training and Gradient Tree Boosting

- The first question we want to ask: what are the parameters of trees?
- what we need to learn are those functions f_k , each containing the structure of the tree and the leaf scores.
- It is intractable to learn all the trees at once.

Additive Training and Gradient Tree Boosting

- Instead, we use an additive strategy: we learn in an iterative manner so we will define the loss function in iteration t as follows:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n}(y_i - \hat{y}_i^{(t)})^2 + \sum_{j=1}^{t}\Omega(f_j)$$

$$= \sum_{i=1}^{n}(y_i - [\hat{y}_i^{(t-1)} + f_t(x_i)])^2 + \sum_{j=1}^{t}\Omega(f_j)$$

$$= \sum_{i=1}^{n}(y_i - \hat{y}_i^{(t-1)} - f_t(x_i))^2 + \sum_{j=1}^{t}\Omega(f_j)$$

$$= \sum_{i=1}^{n}(r_i^{(t-1)} - f_t(x_i))^2 + \sum_{j=1}^{t}\Omega(f_j)$$

Additive Training

- We use an additive strategy: fix what we have learned, and add one new tree at a time.
- We write the prediction value at step t as $\hat{y}_i$.
- Then we have:

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + f_1(x_i) = f_1(x_i)$$

$$\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + f_2(x_i) = f_1(x_i) + f_2(x_i)$$

$$\vdots$$

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) = \sum_{k=1}^{t} f_k(x_i)$$

Additive Training for quadratic loss

- It remains to ask: which tree do we want at each step?
- A natural thing is to add the one that optimizes our objective.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} (y_i - \hat{y}_i^{(t-1)} - f_t(x_i))^2 + \sum_{j=1}^{t} \Omega(f_j) =$$

$$= \sum_{i=1}^{n} [2(\hat{y}^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant$$

Additive Training for general loss function

- The objective function for the general loss function $l(\hat{y}_i, y_i)$ is:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

- If we expand this using second order Taylor expansion ($f(x_0)$ at point x_0):

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + (x - x_0)^2 \frac{f''(x_0)}{2}$$

Additive Training for general loss function

- We expand the function:

$$f(\hat{y}_i^{(t-1)} + f_t(x_i)) = l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$$

at point:

$$\hat{y}_i^{(t-1)}$$

- we get:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

- Where $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)})}$ and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial^2 \hat{y}_i^{(t-1)})}$

Additive Training for general loss function

- After we remove all the constants we remain with:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t)$$

- If we plug in:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

- we get:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Additive Training for general loss function

- We can plug in $f_t(x_i) = w_{q(x_i)}$ and write this expression a bit different:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[g_i f_t(x_i) + h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}[g_i w_{q_{(}x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$\mathcal{L}^{(t)} \approx \sum_{j=1}^{T}(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

- Where $I_j = \{i : q(x_i) = j\}$.

Additive Training for general loss function

- If we further write $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$ we get:

$$\mathcal{L}^{(t)} \approx \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

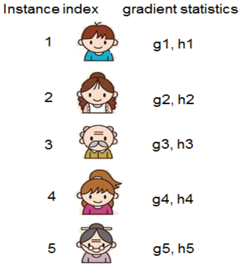
- The form $[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2]$ is quadratic and the minimum $(x = \frac{-b}{a})$ is:

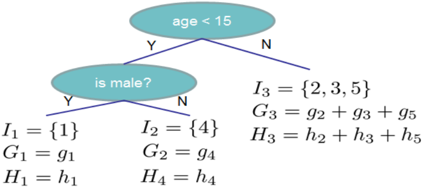
$$w_j^* = \frac{G_j}{H_j + \lambda}$$

- and

$$\mathcal{L}^{*(t)} = -\frac{1}{2} \sum_{t=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Additive Training for general loss function





$$Obj = -\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

The smaller the score is, the better the structure is

SHAP values

- The goal of SHAP is to explain the prediction at a point x by computing the contribution of each feature to the prediction.
- The SHAP method is an idea based on game theory.
- The feature values of a data instance act as players in a coalition.
- Shapley values tell us how to fairly distribute the “payout” (= the prediction) among the features.

SHAP values - main idea

- let $x = \{x_1, x_2, ..., x_p\}$ a particular observation.
- We wish to calculate for each x a function:

$$\hat{f}(x) = \phi_0^x + \sum_{j=1}^{p} \phi_j^x$$

- In other words, we wish to approximate a linear function where the contributions of the predictors add up to the final prediction of point x.
- We will define $\phi_0^x = E_X[\hat{f}(x)]$ so that:

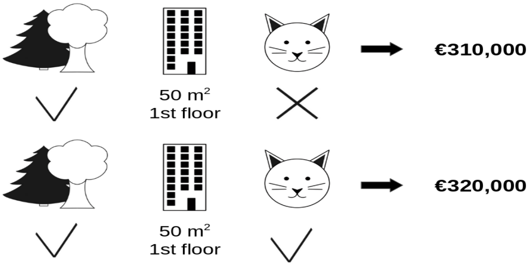
$$\hat{f}(x) = E_X[\hat{f}(x)] + \sum_{j=1}^{p} \phi_j^x.$$

ShAP Values - method for computing feature j SHAP value.

- Step 1: Have a trained machine learning model and a dataset with features and corresponding target values.
- Step 2: Choose a specific point or observation from the dataset for which you want to explain the model's prediction.
- Step 3: Generate Permutations - Create permutations of features, considering different combinations and orders while keeping other features (and j) constant.
- Step 4: Evaluate Model Predictions - For each permutation, use the model to make predictions and observe how the predictions change when once the value of j is not permuted and once when value j is also permuted.

SHAP Values - example

- Assume the following scenario: You have trained a machine learning model to predict apartment prices.
- For a certain apartment it predicts €300,000 and you need to explain this prediction.
- The apartment has an area of 50 m^2, is located on the 2^{nd} floor, has a park nearby and cats are banned:



SHAP Values - example

- No feature values
- park-nearby
- area-50
- floor-2nd
- park-nearby+area-50
- park-nearby+floor-2nd
- area-50+floor-2nd
- park-nearby+area-50+floor-2nd.

SHAP Values - example