理解 XGBoost 和 Light GBM 並實際操作

作者
高念慈、曾宇謙、施永鴻

指導
徐浩雲 助教

May 30, 2022

Abstract

從 Entropy、Gini impurity 出發，再到 Decision Tree ，之後深入 XGBoost 和 Light GBM 原理，最後從 Kaggle 中，選了2021年9月，一份有關於保險的資料，以 XGBoost 和 Light GBM 預測資料，並藉由 Logistic、Decision Tree 和 Random Forest 等模型，相互比較結果。

1 介紹

機器學習是人工智慧的一個分支，涉及機率論、統計學、逼近論、凸分析、計算複雜性理論等多門學科，已廣泛應用於資料探勘、電腦視覺、自然語言處理、生物特徵辨識、搜尋引擎、醫學診斷、檢測信用卡欺詐、證券市場分析、DNA序列定序、語音和手寫辨識、戰略遊戲和機器人等領域。[3]

21世紀，機器學習又一次被人們關注，而這些關注的背後是因爲整個環境的改變，我們的數據量越來越多，硬件越來越強悍，急需要解放人們的生產力，自動去尋找數據的規律，以解決更多專業領域的問題；面對各種機器學習算法，在這裡我們挑選了基於 Boosting 爲框架的主流集成算法 : XGBoost 和 Light GBM。

XGBoost 是大規模並行 boosting tree 的工具，它是目前最快最好的開源 boosting tree 工具包，比常見的工具包快10倍以上； XGBoost 和 GBDT 兩者都是 boosting 方法，除了工程實現、解決問題上的一些差異外，最大的不同就是目標函數的定義，而 Light GBM 主要用於解決 GDBT 在海量數據中遇到的問題，以便其可以更好更快地用於工業實踐中，從名字中我們可以看出其是輕量級（Light）的梯度提昇機（GBM），其相對 XGBoost 具有訓練速度快、內存佔用低的特點。[4]

2 相關工作

我們知道 XGBoost 的基模型不僅支持決策樹，還支持線性模型，不過下面主要爲基於決策樹的目標函數。

Decision Tree :

分離散、連續問題，CART 對回歸樹採用 MSE ，對分類樹採 Gini impurity ，生成結構簡潔的二叉樹；除了 Gini impurity 外， Entropy 也是決策樹常用決定根節點的依據。

Entropy 跟 Gini impurity 都是越低，節點的同質性越高，純節點（相同類）時爲零。

Entropy :

$$H(X) = E[I(X)] = E[-ln(P(X))]$$

有限樣本 : $H(X) = \sum_{i}^{n} P(x_i)I(x_i)$

$$= -\sum_{i}^{n} P(x_i)log_b P(x_i)$$

P : X 的 pmf，b : 底數，決定單位，

I(X) : 信息本體，爲X的資訊量，爲隨機變數

信息增益最大原則 : 選 $Max(IG(T,a))$ 爲根結點

$$Information\ Gain = IG(T,a) = H(T) - H(T|a)$$

ID3、C4.5 演算法都是以信息增益下去遞迴處理

Gini impurity :

$$I_G(p) = \sum_{i=1}^{J} (p_i \sum_{k \neq i} p_k) = \sum_{i=1}^{J} p_i(1 - p_i)$$
$$= \sum_{i=1}^{J} (p_i - p_i^2) = 1 - \sum_{i=1}^{J} p_i^2$$

XGBoost :

eXtreme Gradient Boosting是一種以 Gradient Boosting 爲基礎，結合 baggaing 與 boosting 特性之演算法。針對損失函數， XGBoost 以二階泰勒展開進行逼近：

$$Obj^t \cong \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i^2 f_t(x_i)] + \Omega(f_t)$$
$$g_i = \partial_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} \ell(y_i, \hat{y}^{(t-1)})$$
$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

其中T爲葉子個數，$\sum_{j=1}^{T} w_j^2$ 爲葉節點值平方的加總以一階導函數 g_i 指引梯度方向，以二階導函數 h_i 指引梯度方向如何變化，相對於 GBDT 的一階展開， XGboost 可以更精準的逼近損失函數。此外， XGboost 在節點分裂時不考慮的缺失值的數值。缺失值數據會被分到左子樹和右子樹分別計算損失，選擇較優的那一側。提供我們一種新的缺失值填補方式：不處理。[2]

LightGBM:

LightGBM, Light Gradient Boosting Machine，較常見的是以下兩種以演算法形式改良 : Gradient Boosting 爲基礎，改善 XGboost，最主要以 (1) Gradient-based One-Side Sampling (GOSS演算法)以部分抽樣加上給予權重 :

$$fact = \frac{1-a}{b}$$

a : 抽的梯度大樣本數比例

b : 抽的梯度小樣本數比例

達到不改變分布且加速的效果。 (2) Histogram optimization 把連續的特徵轉換成離散的特徵，使計算速度增快，這兩個演算法改良 XGboost 的預排序演算法。

除了演算法的改良，LightGBM 也提供幾種樹的優化，包含 (1) 帶深度限制的 leaf wise 算法， LightGBM 使用按葉生長，不使用 XGboost 的 level wise算法。 (2) 類別特徵最優分割，我們不使用 one hot 編碼,避免稀疏矩陣的存在，浪費過多時間。[1]

ROC :

Receiver operator characteristic (ROC) 是以混淆矩陣作爲基礎，給定閾值,就可以畫出各個閾值的 ROC 曲線圖。而 ROC 曲線圖 X 軸代表僞陽性率，Y 軸代表眞陽性率，這樣只要 ROC在 X = Y 這條直線之上就可以得知是一個好的模型，反之在 X = Y 這條線之下就可以得知是一個差的模型。

一般來說 ROC 曲線很容易可以看出任意閾值對學習器的好壞,而閾值簡單來說就是臨界值，超過或等於臨界值就判定爲陽性，未超過則判斷爲陰性，配合混淆矩陣的定義，即可繪製出 X 軸(僞陽性率, FP)、Y 軸(眞陽性率, TP)，乍看之下 ROC 曲線是相當簡單、容易的方式，但卻有個致命的缺點:若兩條 ROC 曲線要比較好壞，光看曲線是無法比較的。

AUC :

Area under the Curve of ROC (AUC) 簡單來說，AUC是 ROC 曲線下的面積，正因爲是 ROC 曲線下的面積，因此 :

$$AUC \in [0, 1]$$

，最主要的用處是我們希望可以比較兩條甚至更多條 ROC 曲線的好壞。至於如何分辨好壞，我們由 ROC 曲線得知，ROC 曲線越靠近 (0,1) 也就是圖左上方就越好，而 AUC 曲線也雷同，越往上方遠離 X = Y 則 AUC 面積越大。

我們如何計算 AUC，AUC是 ROC 曲線下的面積，既然都說曲線，第一個想法是積分，但很快就知道行不通，因爲 ROC 是以各種不同的各種不同的閾值去描繪出來的曲線，因此有可能不是某一個函數，所以使用的是近似方法中的梯形法，梯形法也就是將每個相鄰的以直線相連，計算他們的總面積，因爲每一個線段下方都是一個梯形，因此稱爲梯形法。而另一種方法是 ROC AUCH 法。

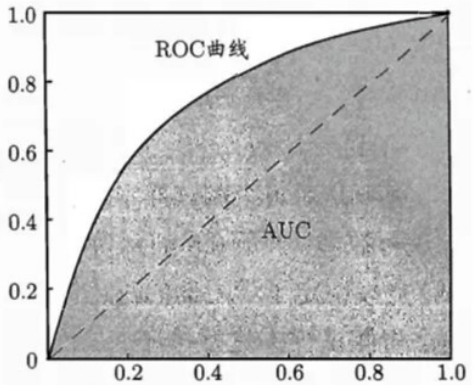

Figure 1: ROC與AUC

在應用的部分，先利用 5-Fold 讓不同分組訓練的結果，進行平均來減少方差，使模型的性能對數據的劃分不那麼敏感，又爲了能讓 5 個模型相互比較，我們統一設置 $random\_state = 42$ 決定分割；而在觀察指標時，考慮到預測輸出爲 0 ∼ 1 的小數，跟基本事實 0 *or* 1 二元分類不好比較，於是保守取了0.5作爲 Confusion matrix 的閾值，並分別算出各模型的準確度，最後藉由 KDE plot 彌補直方圖長條寬度上的缺陷，以高斯核賦予預測輸出平滑性讓我們好觀察模型分布。

3 應用

3.1 資料

我們使用 Kaggle 上一份有關保單的資料進行實作(https://www.kaggle.com/competitions/tabular-playground-series-sep-2021/overview)。目標是預測客户是否根據保險單提出索賠。0 ∈索賠，1 ∈不索賠。但預測可以是從 0.0 到 1.0 的任何數字，表示索賠的概率。此數據集中的特徵已匿名化，可能包含缺失值。

(1) 訓練集：共 957919 人，118 個特徵，特徵彼此相關性低，分佈亦不相同，且有缺失值隨機分佈在各個特徵，佔整體資料的 1.6%。

(2) 測試集：性質與訓練集相似，實作目標爲預測 493000 人的索賠機率。

(3) 缺失值處理： 這是一份有關於保單的資料，在保險行業有二特別性質，

道德風險：在購買汽車的保險後，有些人可能就不會鎖上汽車，反而增加車子被偷危險性。

逆選擇：高疾病風險者更願意進行投保。
客户爲了更順利的提出索賠，可能故意隱藏對自身不利的信息 – 缺失值的存在對於索賠具有意義，需要被納入考量。

3.2 EDA

(1) 存在缺失值？佔整體 1.6%，均勻分散

(2) 各特徵缺失值比例：每個特徵皆缺失1.6%

(3) 目標變數是否不平衡：否

(4) 統計性質：各值域有明顯差距

(5) Box plot 、分布圖：各分布都差異頗大

(6) 考慮新特徵：以每位顧客Na數作爲新特徵

(7) 相關性：除了Na數外，其於約爲0.03上下

(8) Claim V.S. 缺失值數量：相關性 0.45

(9) 補缺失值：由不做處理搭配新特徵勝出

3.3 模型

5 個模型皆沒有常態假設，在用 Scikit-learn 訓練時因爲 Logistic 、 Decision Tree 、 Random Forest 不能接受 NaN ，故將缺失值統一補 0。

3.3.1 Logistic

$$Logistic(f(x)) = \frac{1}{1 + e^{-}f(x)}$$

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ ...$$

因爲 β_i 大小會直接影響預測機率，所以事前對資料做標準化，讓各特徵貢獻(尺度)相同。

參數上面主要採用原預設值：

(1) fit_intercept : True

(2) max_iter =10000

(3) penalty : 'l2'

(4) solver : 'lbfgs'

(5) tol : 0.0001

3.3.2 Decision Tree

每個特徵單獨處理(*Entropy or Gini impurity*)，幾乎不需要數據預處理。

主要參數 :

(1) criterion = "gini"

(2.1) max_depth = 5

(2.2) max_depth = 60

3.3.3 Random Forest

由 *Bagging* + 隨機特徵採樣的多個決策樹組成，所以同決策樹，不須標準化。

主要參數 :

(1) max_features = 'sqrt'

(2) max_samples = None

(3) criterion = 'gini'

(4) bootstrap = True

(5.1) max_depth = 60

(5.2) max_depth = 5

3.3.4 XGBoost

XGboost 演算法在未調參時，其 *roc_auc_score* 已與前三種模型不分伯仲，足以見該演算法的強大之處。在調參的過程中，我們更重視“避免過擬合”這一概念的落實與學習時間的縮減 :

(1) Learing_rate : 0.1，該設置越低，對訓練集的學習越深，花費時間越久

(2) n_estimators : 3000，最多生成 3000 顆決策樹，用以配合 *Learing_rate* 的學習率

(3) Max_depth : 5，單一決策樹的最大深度，設置過大可能會過擬合

(4) min_child_weight : 75，如一節點 $\sum_{i=1}^{n} h_i$<75，即停止分裂，防止模型學到局部樣本，防止過擬合

(5) gamma : 0.1，如一節點分裂後損失函數減少程度 >0.1，該節點才分裂，設置越大越保守

(6) subsample : 0.55，對於每棵樹，只從訓練集中隨機抽 0.55% 做訓練，避免過擬合

(7) reg_alpha : 10，權重的 L1 正化，使算法更快

(8) random_state : 42，用以複製隨機數據的結果，確保 5-FOLD 切割位置相同

3.3.5 Lightgbm

Lightgbm 視爲 XGboost 的改良，其大部分參數與 XGboost 具相同意義 :

(1) Learing_rate : 0.095

(2) reg_alpha : 25

(3) Max_depth : 4

(4) subsample : 0.5

(5) n_estimators : 10000

(6) random_state : 42

亦有部分相異 :

(7) Max_bin : 200， Lightgbm 特有的直方圖演算法，將 200 個連續值轉爲離散以減少內存與執行時間

(8) num_leave : 10， Lightgbm 允許左右節點分裂深度不相同，一邊最大深度大致爲 Max_depth 的兩倍

(9) reg_lambda : 17，權重的 L2 正則化項，減少過擬合

3.4 Submission 調整

單一模型有其優劣，如 Logistic 只關注特徵與預測的線性關係，而忽略其他。但只看線性關係的話，Lightgbm 不一定表現的比 Logistic 更好。故我們使用集成學習的方式，結合各個模型的預測，期望能提升我們最終預測的準確率 :

(1) Voting : 各個模型預測分權加總

$$\frac{2}{5}\,XGB\,+\,\frac{2}{5}\,LGB\,+\,\frac{1}{15}\,Logistic\,+\,\frac{1}{15}\,Decision\,Tree\,+\,\frac{1}{15}\,Random\,Forest$$

(2) Power averaging : Voting 的係數取決於設計者的主觀認定，而 Power averaging 提供較規範的設計 :

Correlation jumping : 依 roc_auc_score 與模型相關度進行排序，

$$LGB{>}XGB{>}Decision\,Tree{>}Random\,Forest{>}Logistic$$

Logistic 與其他四個模型相關性過低，不考量。

Threshold power : 計算等式

$cov(LGB, Random\ Forest)^n = 0.9\ \rightarrow\ n = 9$
$n > 9$，會有方差增加的危險，$n < 9$會有偏差增加的危險。

Ensembling :

$$\frac{(\frac{9\,LGB\,+\,9XGB}{2}\,+\,\frac{9\,Decision\,Tree\,+\,9\,Random\,Forest}{2})}{2}$$

4 討論

模型表現	LGB	XGB	Decision Tree	Random Forest	Logistic	Kaggle 第一名
模型時間	1.87h	4.07h	38.8min	4.06h	54sec	1.53h
準確率	0.774	0.772	0.772	0.772	0.736	不詳
kaggle得分	0.81687	0.81462	0.80239	0.80322	0.7989	0.81875

Table 1: 集成學習

集成學習	Voting	Power averaging
選取模型	五個	四個
係數決定	自由選取	根據 Threshold power
kaggle得分	0.81621	0.81465

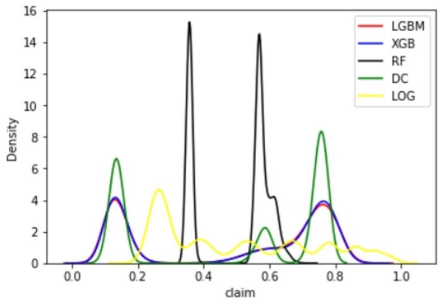

Figure 2: KDE plot

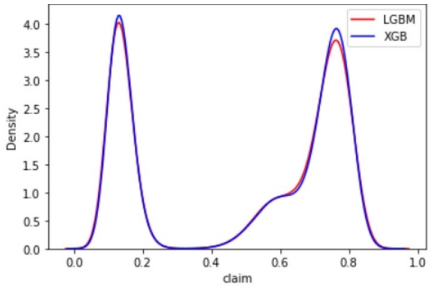

Figure 3: XGboost & LightGBM kde plot

模型比較	XGboost	LightGBM
演算法	預排序演算法	直方圖 & GOSS演算法
決策數生長方式	Level wise	Leaf wise
類別特徵優化	不支持類別特徵	支持類別特徵
使用樣本	全樣本	部分小梯度樣本

5 結論

可以發現 Logistic 在資料筆數筆數將近 100 萬外加使用 5-fold 下，不需要 1 分鐘就能有大概 0.79 的成績；而 Decision Tree 則是很明顯的在 max_depth = 60 時，出現過擬合的現象，分數從 max_depth = 5 時的約 0.800 掉到了約 0.644，時間更是從 9 分內上升到 45 分鐘；不過過擬合的現象就沒出現在 Random Forest 上了，不管是 max_depth = 5 還是 max_depth = 60，兩者分數都差不多落在 0.799 上下，正好證實了 Bagging 有降低過擬合的功能；雖然 max_depth = 60 的評分皆高於 max_depth = 5，不過在考慮到時間花費上，max_depth = 60 的 4 個小時，跟 max_depth = 5 的 38 分鐘，未來可能會較優先考慮 max_depth 較小的情形。

因應不同的資料特性，應該使用不同的方式處理特徵，在本篇實作中，以缺失值個數作爲新特徵比填補更爲有效。由模型表現可知，Lightgbm 在一定速率下，有最優的 kaggle 得分。在本次實作中爲最強大的模型。符合其定位：“XGboost 的優化演算法”。

Voting 之所以沒辦法提升準確率，我們有以下猜想：

(1) 加權係數設置不得當

(2) Ligthgbm 的表現全面碾壓其他模型，引入其他模型進行 Voting 反而會惡化結果。

而 Power averaging，其主旨是藉由高度相關的多個模型集合以提升準確率，在本次實作中，Logistic 與其餘四個模型關聯較少，且總使用模型數只有 4 個，進而導致集成分數遠不如單一模型。

References

[1] Thomas Finley3 Taifeng Wang1 Wei Chen1 Weidong Ma1 Qiwei Ye1 Tie-Yan Liu Guolin Ke1, Qi Meng2. Lightgbm: A highly efficient gradient boosting decision tree. [4]. https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf Accessed 16 June 2017.

[2] Carlos Guestrin Tianqi Chen. Xgboost: A scalable tree boosting system. [3]. https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf Accessed March 27, 2016.

[3] wikipedia. . [1]. https://zh.wikipedia.org/zh-tw/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0 Accessed January 26, 2022.

[4] . ——xgboostlightgbm. [2]. https://zhuanlan.zhihu.com/p/87885678 Accessed November 1, 2019.