



# python で データ解析

7. 第 5 ~ 6 回 [XGBoost] のまとめ +  $\alpha$

## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回：2 値データの分類問題 [Titanic]
- 第 6 回：回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

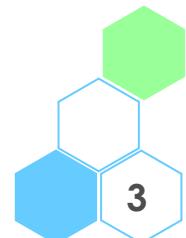
## kaggle: <https://www.kaggle.com/>

- 企業や研究者がデータを投稿し、世界中の統計家やデータ分析その最適なモデルをコンペという形で競い合う、予測モデリング及び分析手法関連のプラットフォーム及びその運営会社 ( Wikipedia より: <https://ja.wikipedia.org/wiki/Kaggle>)
- 初心者向けのチュートリアルや上級者のプログラム公開、Discussion 等があり、コンペに参加しなくても非常に勉強になる
- 次頁の House Prices データは kaggle の学習用コンペのもの、kaggle に登録(無料)すればダウンロードすることが出来る

The screenshot shows the 'Competitions' section of the Kaggle website. On the left, there's a vertical sidebar with icons for navigation. The main area displays two active competitions:

- Titanic: Machine Learning from Disaster**: Predict survival on the Titanic and get familiar with ML basics. Status: Getting Started • Ongoing • 17246 Teams. Category: Knowledge.
- House Prices: Advanced Regression Techniques**: Predict sales prices and practice feature engineering, RFs, and gradient boosting. Status: Getting Started • Ongoing • 4440 Teams. Category: Knowledge.

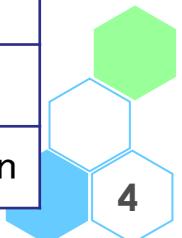
Below these, there's a section for 'All Competitions' with filters for Active (Not Entered), Completed, InClass, All Categories, and Default Sort.



## 第5回: *Titanic*

- [kaggle](#) のコンペで使用されている、[タイタニック号に関するデータ](#)
- train.csv(学習用データ)と test.csv(テストデータ)がある

変数	定義	中身の補足
PassengerId	ID	train.csv: 1～891、test.csv: 892～1309
<b>Survived</b>	<b>生死 → 目的変数</b>	<b>0 = No, 1 = Yes: test.csv にはなし</b>
Pclass	乗車券のクラス	≒経済状況: 1=1st, 2=2nd, 3=3rd
Name	名前	例: "Heikkinen, Miss. Laina"、敬称付き
Sex	性別	male, female
Age	年齢	
SibSp	同乗している兄弟や配偶者の数	
Parch	同乗している親や子供の数	
Ticket	乗車券番号	例: PC 17599
Fare	料金	
Cabin	客室	例: C85
Embarked	乗船港	C = Cherbourg, Q = Queenstown, S = Southampton



# 第6回: House Prices

- [kaggle](#) のコンペで使用されている、[住宅価格に関するデータ](#)
  - アイオワ(Iowa)州エイムズ(Ames)における、住宅のほぼ全ての要素を説明する 79 個の変数より、各住宅の最終価格(SalePrice)を予測することが目的
- train.csv(学習用データ)と test.csv(テストデータ)がある

変数	定義	中身の補足
Id	ID	train.csv: 1~1460、test.csv: 1461~2919
SalePrice	住宅価格(ドル)	→ 目的変数
その他 79 変数、欠測多数、評価関数は <a href="#">Root MSE</a> 、 <a href="#">詳細はこちら</a>		

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	...	YrSold	SaleType	SaleCondition	SalePrice
1	60	RL	65	8450	Pave	...	2008	WD	Normal	208500
2	20	RL	80	9600	Pave	...	2007	WD	Normal	181500
3	60	RL	68	11250	Pave	...	2008	WD	Normal	223500
4	70	RL	60	9550	Pave	...	2006	WD	Abnorml	140000
5	60	RL	84	14260	Pave	...	2008	WD	Normal	250000
:	:	:	:	:	:	...	:	:	:	:
1460	20	RL	75	9937	Pave	...	2008	WD	Normal	147500

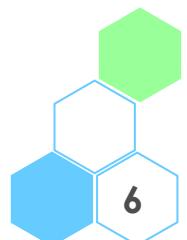


# XGBoost の概要

- XGBoost: Extreme Gradient Boosting の略称で、学習データを使用する教師あり学習の手法、**GBDT**(Gradient Boosting Decision Tree)の 1 つ
  - 勾配降下法(Gradient): 目的変数と予測値から計算される「目的関数」の最小化問題、例えば誤差平方和(凸関数)を最小にするパラメータ値の探索に適用される
  - ブースティング(Boosting): アンサンブル学習器の一種で、学習器を直列的に複数組み合わせて精度を高くする手法、各学習器は精度が低くても良い
  - 決定木(Decision Tree): 学習器の 1 つ、精度は一般的に低い
- $n$  個のデータ、 $m$  個の説明変数からなるデータセット  $D = \{(x_i, y_i)\}$   
 $(|D| = n, x_i \in R^m, y \in R)$  に対して、以下の  $K$  個の関数の和により予測を行う( $K$  個の木により予測を行う)アンサンブルモデルを考える
  - $\hat{y}_i = \phi(x_i) = \sum_{k=1}^K f_k(x_i), f_k \in F, i = 1, \dots, n$ 
    - $F = \{f(x) = w_{q(x)}\}$  ( $q: R^m \rightarrow R^T, w \in R$ ) は取りうる決定木(CART)の空間
    - $q$  は  $n$  個の各データを、対応する各葉に紐づけする「木の構造」
    - $T$  は木の葉の数、 $f_k$  は独立した木の構造  $q$  と葉の重み  $w$  に対応する
    - $w_i$  を使用して、 $i$  番目の葉(node:ノード)のスコアを表す

※ 出典:Tianqi Chen and Carlos Guestrin (2016) "[XGBoost: A Scalable Tree Boosting System](#)"

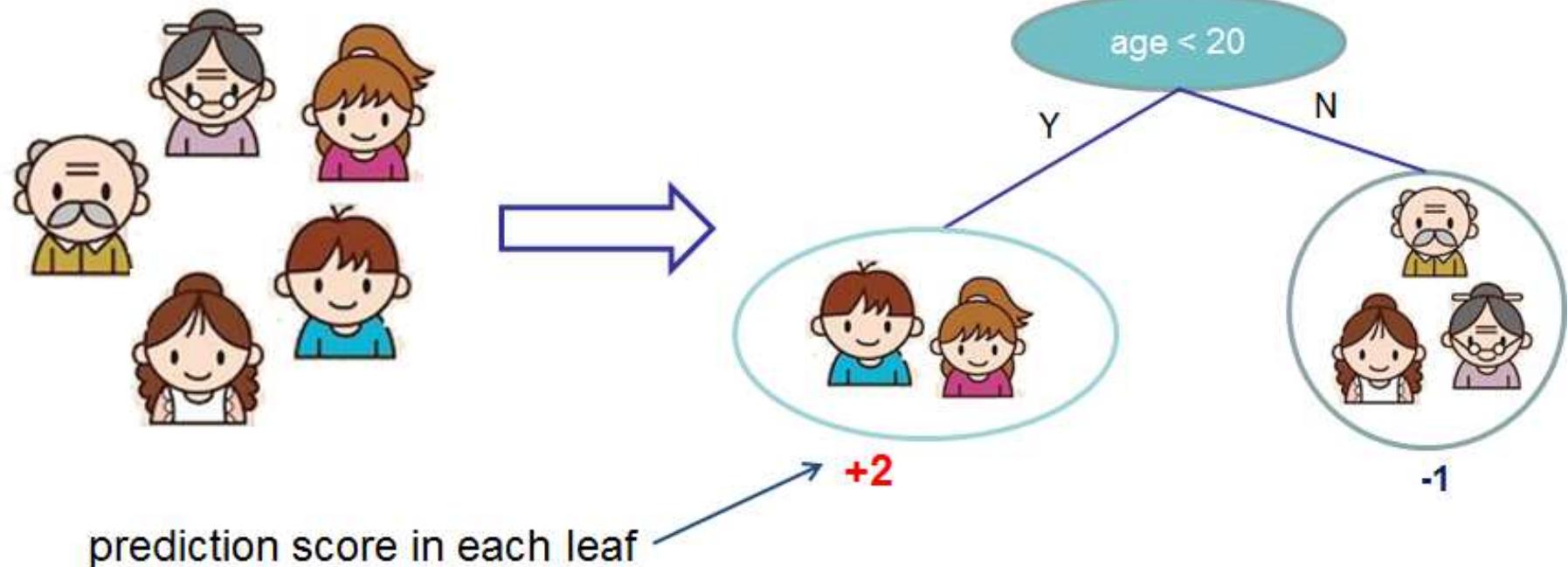
XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>



## XGBoost の概要: XGBoost Documentation の例

Input: age, gender, occupation, ...

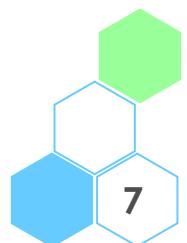
Like the computer game X



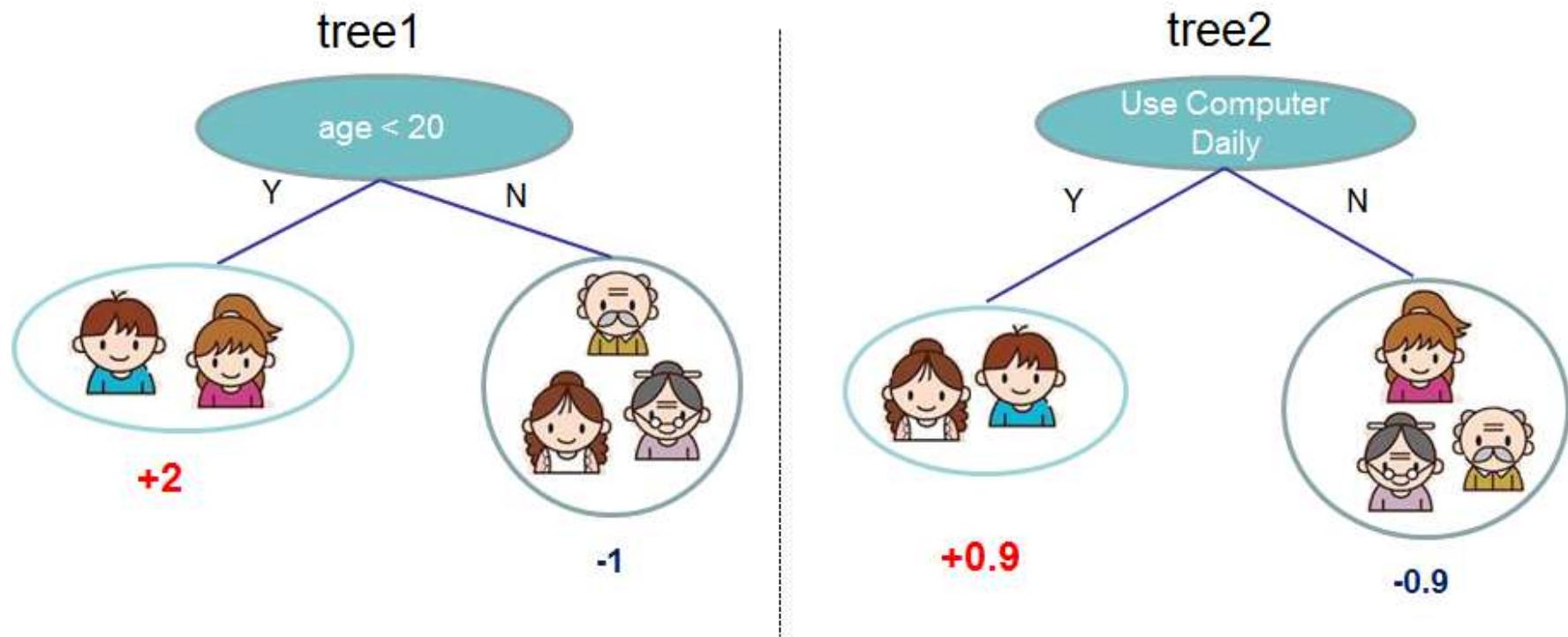
- 5人の家族の「コンピュータゲームの好みの度合い(連続値)」を予測する
- CART では、家族のメンバーを様々な葉に分類し、対応する葉にスコアを割り当てるため、実際のスコアが各葉に関連付けられる
- ただし、1本の CART の木では十分な予測精度が見込めない

※ 出典: Tianqi Chen and Carlos Guestrin (2016) "[XGBoost: A Scalable Tree Boosting System](#)"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>



# XGBoost の概要: 論文の例(図 1)



$$f(\text{boy}) = 2 + 0.9 = 2.9$$



$$f(\text{elderly man}) = -1 - 0.9 = -1.9$$

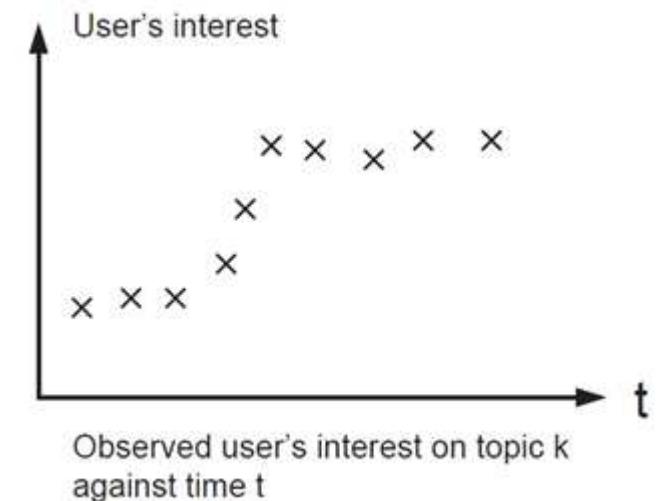
- q で与えられる木の決定ルールを使用してそれを葉に分類し、対応する葉のスコア (w で与えられる) を合計することにより、最終的な予測を計算する  
⇒ アンサンブル学習(のうち、XGBoost はブースティングという手法)

※ 出典: Tianqi Chen and Carlos Guestrin (2016) "XGBoost: A Scalable Tree Boosting System"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

# XGBoost の概要

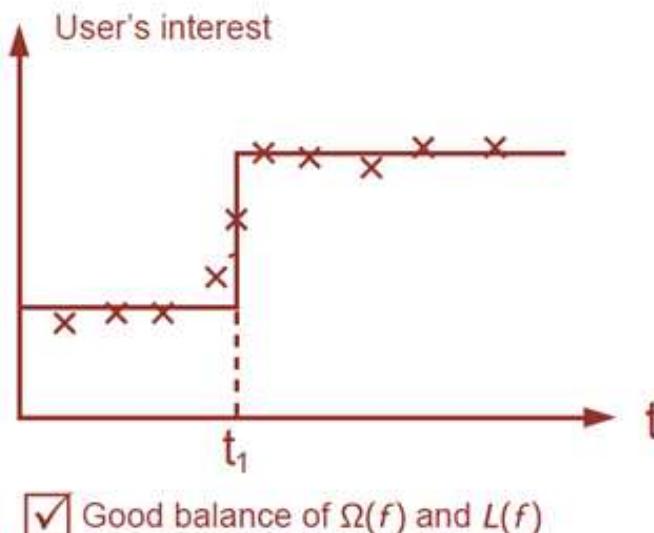
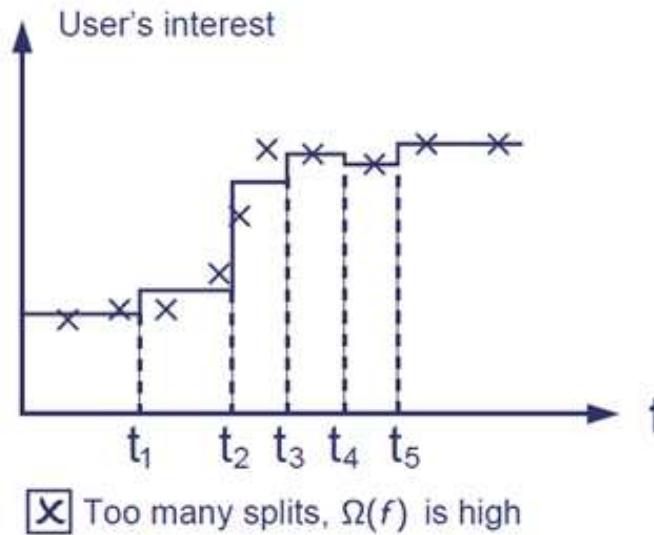
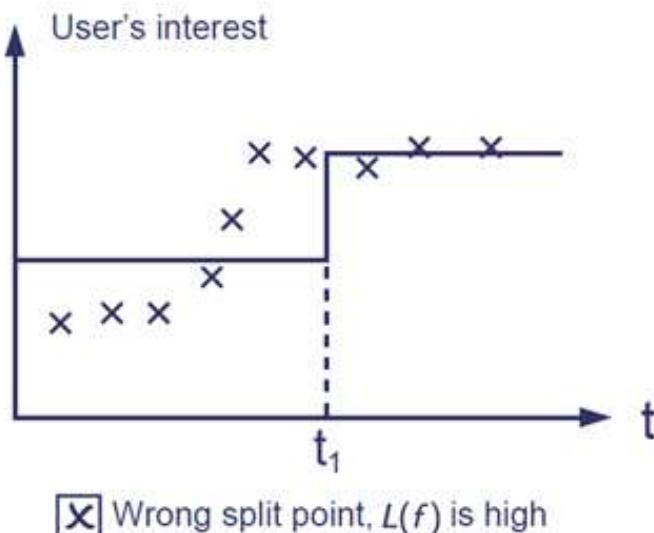
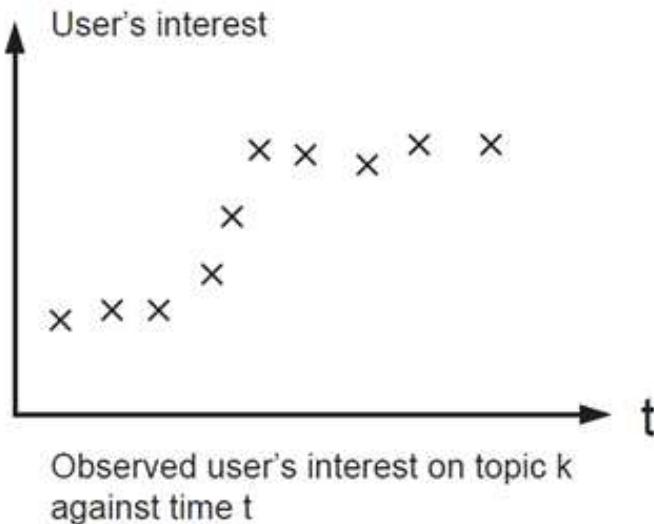
- 以下の目的関数  $L(\phi)$  を最小化することで、モデルを学習させる
  - $L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$
  - $\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$  (実装ではさらに  $L_1$  正則化項  $\alpha \|w\|$  が追加)
    - $l$  は微分可能な損失関数(凸関数)で、目的変数の値と予測値との差  
(例: 分類問題の場合には logloss、回帰問題の場合には平均二乗誤差)
    - $\Omega(\cdot)$  は木の複雑度を表す正則化項(罰則項)で、 $\gamma$  にて葉(ノード)の数、 $\alpha$  にて  $L_1$  正則化項、 $\lambda$  にて  $L_2$  正則化項の強さの調整を行う
- 正則化項はモデルの複雑さを制御し、モデルが(学習)データへ過剰に適合しないために用いられる
- 例として、右図のデータに階段関数を当てはめることを考える



※ 出典: Tianqi Chen and Carlos Guestrin (2016) "XGBoost: A Scalable Tree Boosting System"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

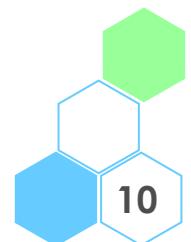
# XGBoost の概要



- 右上図: 複雑すぎ
- 左下図: 単純だが不適切
- 右下図: 適切

## 【一般的な原則】

- 単純なモデルと予測モデルの両方が必要である
- この2つの関係は機械学習におけるバイアスと分散のトレードオフとも呼ばれる



※ 出典: Tianqi Chen and Carlos Guestrin (2016) "XGBoost: A Scalable Tree Boosting System"

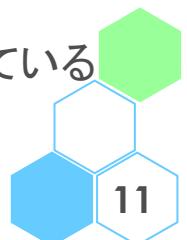
XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

# XGBoost の概要

- XGBoost では additive manner と呼ばれる方針でモデルが学習される
  - まず、特定の木の構造  $q$  を与えた上で最適な重み  $w$  を算出し、次に算出された最適な重みより、木の構造としてどれが最も良いかを目的関数  $L^{(t)}$  の値から判定
- $\hat{y}_i^{(t)}$  を  $t$  回目のくり返しにおける  $i$  番目のデータの予測値とすると、
  - $\hat{y}_i^{(0)} = 0$
  - $\hat{y}_i^{(1)} = f_1(\mathbf{x}_i) = \hat{y}_i^{(0)} + f_1(\mathbf{x}_i)$
  - $\hat{y}_i^{(2)} = f_1(\mathbf{x}_i) + f_2(\mathbf{x}_i) = \hat{y}_i^{(1)} + f_2(\mathbf{x}_i)$
  - $\dots$
  - $\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(\mathbf{x}_i) = \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)$
- $t$  回目のくり返しにおける木の構造を決めるため、以下の関数を考える
  - $L^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\right) + \Omega(f_t)$ 
    - 最適化の対象は  $f_t(\mathbf{x}_i)$  なので、 $\sum_{i=1}^t \Omega(f_i)$  の  $1, \dots, t-1$  に関する項は削除している

※ 出典: Tianqi Chen and Carlos Guestrin (2016) "[XGBoost: A Scalable Tree Boosting System](#)"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>



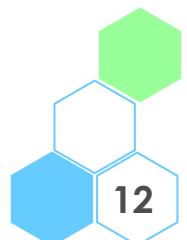
# XGBoost の概要

- $l(\cdot)$  から  $f_t(x_i)$  を外に出すため、 $L^{(t)}$  を 2 次までの項で近似し
  - $L^{(t)} \cong \sum_{i=1}^n \left[ l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$
  - $g_i = \partial_{\hat{y}^{(t-1)}} \left( y_i, \hat{y}_i^{(t-1)} \right)$ ,  $h_i = \partial_{\hat{y}^{(t-1)}}^2 \left( y_i, \hat{y}_i^{(t-1)} \right)$
- 定数項は考えなくてよいので削除し、 $t$  回目のくり返しにおいて、
  - $\tilde{L}^{(t)} \cong \sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$ 
    - この式が、新しい木を追加した場合の最適化の目的関数になる
    - この定義の重要なポイントは、目的関数の値が  $g_i$  と  $h_i$  のみに依存する点
    - XGBoost が様々な損失関数をサポートする理由で、**様々な損失関数を、全く同じ方法を使用して最適化できることになる**

$$\begin{aligned} \bullet \quad & l(\cdot) \text{ が MSE の場合、} L^{(t)} = \sum_{i=1}^n \left\{ y_i - \left( \hat{y}_i^{(t-1)} + f_t(x_i) \right) \right\}^2 + \Omega(f_t) \\ & = \sum_{i=1}^n \left\{ 2 \left( \hat{y}_i^{(t-1)} - y_i \right) f_t(x_i) + f_t(x_i)^2 \right\} + \Omega(f_t) + \text{定数項} \end{aligned}$$

※ 出典: Tianqi Chen and Carlos Guestrin (2016) "[XGBoost: A Scalable Tree Boosting System](#)"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

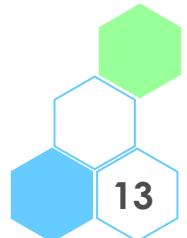


# XGBoost の概要

- 正則化項  $\Omega(f_t)$  を書き下し、
  - $\tilde{L}^{(t)} \cong \sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$   
 $= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T$
  - $I_j = \{i | q(x_i) = j\}$ : 「 $j$  番目の葉に割り当てられたデータ」の添字の集合  
( instance set )
  - データの方向の総和  $\Sigma_i$  を、木のノードの方向の総和  $\Sigma_j$  に書き換えている
- $q(x)$  は固定されているとすると、 $j$  番目の葉における最適な重み  $w_j^*$  は
  - $w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$
  - となり、その際の最適な目的関数の値が以下のように求まる
  - $\tilde{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^T \frac{\left( \sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$

※ 出典: Tianqi Chen and Carlos Guestrin (2016) "[XGBoost: A Scalable Tree Boosting System](#)"

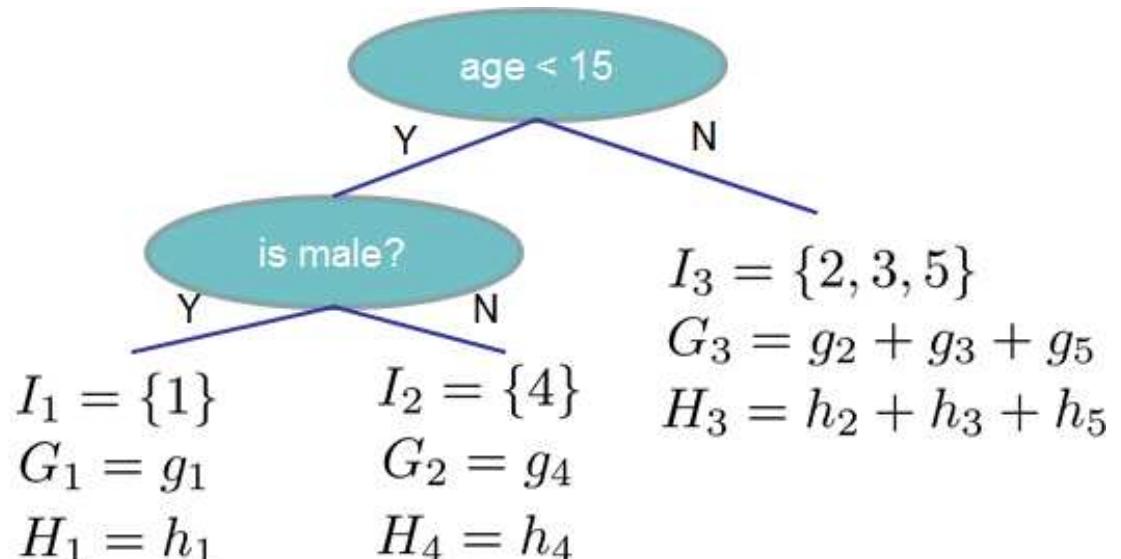
XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>



# XGBoost の概要

- 前回の  $\tilde{L}^{(t)}(q)$  を、木構造  $q(x)$  の「良さ」を測るスコアとして使用する
- $t$  回目のくり返しにおいて、木がどれだけ優れているかを測定する方法が計算できたので、「考えられるすべての木を列挙して最適な木を選択」出来れば理想的だが非現実的…

Instance index	gradient statistics
1	 g1, h1
2	 g2, h2
3	 g3, h3
4	 g4, h4
5	 g5, h5



$$\tilde{L}^{(t)}(q) = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

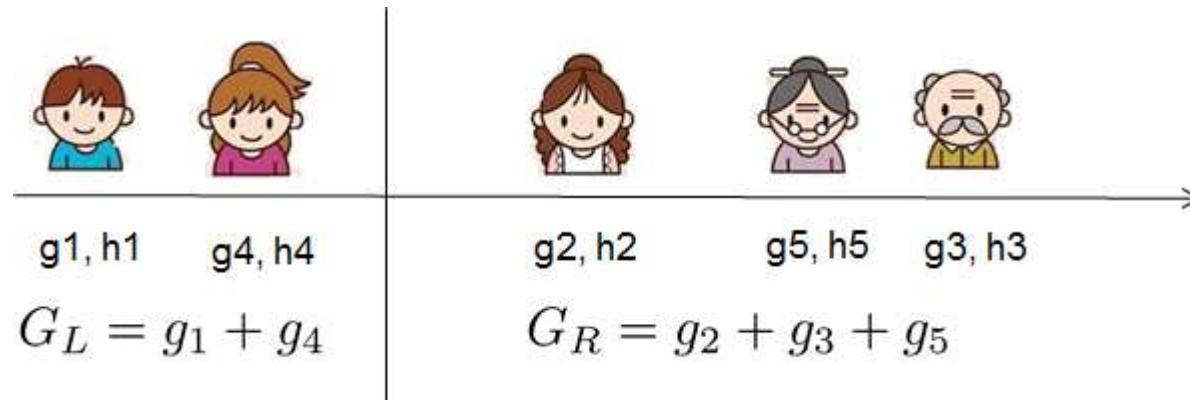
The smaller the score is, the better the structure is

※ 出典:Tianqi Chen and Carlos Guestrin (2016) "XGBoost: A Scalable Tree Boosting System"

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

# XGBoost の概要

- XGBoost では「仮に葉を 2 つの葉に分割しようとする際に得られる減少スコア」を以下で計算し、木を分割するかどうかを判定する(  $L_{split}$  が正であれば分割)
  - $$L_{split} = \frac{1}{2} \left[ \frac{\left( \sum_{i \in I_L} g_i \right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left( \sum_{i \in I_R} g_i \right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left( \sum_{i \in I} g_i \right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$
    - $I_L$ : 分割後の左の葉に割り当てられたデータの添字の集合( instance set )
    - $I_R$ : 分割後の右の葉に割り当てられたデータの添字の集合( instance set )
    - $I = I_L \cup I_R$
- データが実数の場合、最適な分割を効率的に検索するために、全てのデータを昇順に配置し、左から順に全ての分割について調べ、最適な分割を見つける



※ 出典:Tianqi Chen and Carlos Guestrin (2016) “[XGBoost: A Scalable Tree Boosting System](#)”

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

# XGBoost の概要: 過学習防止の方策

## Shrinkage

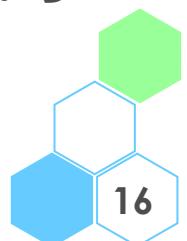
- 新しい木が生成された際、出力  $f_t(x_i)$  に学習率  $\eta$  ( $0 < \eta < 1$ ) をかけ、値を縮小することで過学習を防止
  - 予測値  $\hat{y}_i^{(t)}$  を算出する際に、各  $f_t(x_i)$  に学習率  $\eta$  を乗算している

## Subsampling

- 学習時の各回( $t$ )にて、全てのデータ／説明変数を用いずに一部を採用することで過学習を防止
  - (Row) subsampling: 木ごとに学習データの行を sub-sampling する
  - Column subsampling by tree: 木ごとに列(説明変数)を sub-sampling する
  - Column subsampling by level: 木のレベル(深さ)ごとに列を sub-sampling する
  - Column subsampling by node: ノード(枝の分割時)ごとに列を sub-sampling する
- 他にも、木の本数(num\_round／n\_estimators、学習率との兼ね合い)、木の深さ(max depth、深いと学習も深まる)、葉を構成する最小データ数(min. child weight、値が大きいと過学習防止)等が調整できる
  - 木の本数は、「木を増やす⇒目的関数のスコアを計算」をくり返し、ある時点から一定回数くり返してもスコアが改善しないところで決定する、等の方法がある  
**⇒ early stopping**

※ 出典:Tianqi Chen and Carlos Guestrin (2016) “[XGBoost: A Scalable Tree Boosting System](#)”

XGBoost Documentation: Introduction to Boosted Trees: <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

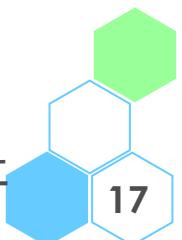


# XGBoost.train() のパラメータ

- **booster**: “gbtree”、“dart” が適している場合もあるが稀
- **objective**: “reg:squarederror”(回帰、平均二乗誤差を最小化)、“reg:squaredlogerror”(回帰、 squared log loss を最小化)、“binary:logistic”( 2 値分類、log lossを最小化、確率を返す)、“count:poisson”( poisson 回帰)、“survival:cox”( Cox 回帰)、“multi:softprob”(多値分類)、他
- **eval\_metric**: デフォルトは objective の手法に応じた指標(回帰:“rmse”、分類:“logloss”)、“rmsle”、“mae”、“error”( 2 値分類:1-accuracy )、“merror”(多値分類)、他
- **num\_round [n\_estimators]**: 決定木の本数
- **eta [learning\_rate]**: 学習率、デフォルトは 0.3 だが、0.05 又は 0.1 あたりが初期値
- **max\_depth**: 決定木の深さ、デフォルトは 6 で、3~9 あたりが初期の候補
- **gamma**: 決定木を分岐させるために最低限減らすべき目的関数の値、値が大きいと分岐しづらい(過学習防止)が、デフォルトは 0
- **min\_child\_weight**: 葉を構成する最小データ数、値が大きいと過学習防止、デフォルトは1で、0.1~10
- **subsample**: 決定木ごとに学習データの行をsamplingする割合、値が小さいと過学習防止、デフォルトは 1 だが、0.6~0.95 あたり
- **colsample\_bytree**: 決定木ごとに特徴量の列をsamplingする割合、値が小さいと過学習防止、デフォルトは 1 で、0.6~1 あたり
- **colsample\_bylevel**: 深さごとに特徴量の列をsamplingする割合、デフォルトは 1 で、0.5~1、0.3 程度に下げる場合も
- **colsample\_bynode**: ノード(分割)ごとに特徴量の列を sampling する割合で、新しい分割が評価されるたびに sampling され、現在の深さ用に選択された列の集合から sub-sampling される
- **alpha [reg\_alpha]**: L1 正則化の強さ、値が大きいと過学習防止、デフォルトは 0
- **lambda [reg\_lambda]**: L2 正則化の強さ、値が大きいと過学習防止、デフォルトは 1
- **seed**: 乱数のシード、**n\_jobs**: スレッド数、-1 でフル稼働

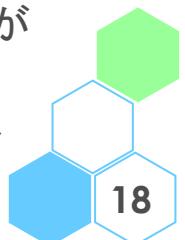
※ XGBRegressor() と XGBClassifier() は、num\_round と eta は [n\_estimators] [learning\_rate] に

<https://xgboost.readthedocs.io/en/latest/parameter.html>



# XGBoost で予測する方針( 2021.3.1 現在)

- 目的は「シンプルでリークを起こしにくく、そこそこ精度の良い方針( kaggle で上位15%以内が望ましい)の探索」、kaggle でメダルを取るためのテクニック、等は追及しない
  - kaggle に参加する理由は、予測精度が測定(リークの有無が判断)出来るため
- 前処理、データの概要調査、データクリーニングは必須
  - 必要に応じて目的変数を対数変換、データの間違い修正、外れ値は 1%・99% 点などで補完
- 変数は数値、データの数値の大きさには意味がなく大小関係のみが影響
  - 巷では「feature engineering」「EDA」の方法は回帰、NN 等を前提とした方法が紹介されていることが多い、XGBoost の様な決定木ベースの場合は別の方針が必要かも
  - スケーリングや標準化・正規分布への変換は不要、相関係数も過度に信用しないこと
- 欠測値があってもそのまま処理できる
  - 变に欠測を埋めた場合よりも、欠測値を放置した方が精度が上がる場合がほとんど
  - 「欠測値を放置」+「欠測であるフラグ」を追加すると精度が上がる場合が多い
  - 連続変数の欠測は放置、カテゴリ変数を encoding する際は one-hot encoding + 欠測もカテゴリのひとつとして変換を行う
  - catboost encoding 等の target encoding はリークが怖いので使用しない
- 決定木ベースの手法では、変数が大量になければ変数選択は重要でない
  - XGBoost の場合、予測に寄与しない変数があっても予測精度にあまり影響しない
  - 人が判断するよりも、モデルに含めることができるならモデルに含めて XGBoost に判断してもらうのが良い? … ということで、今回は原則として全変数を使う
- 新たな変数作成は必要に応じて(変数内での集約、変数間の計算)
  - 第 5 回 : Titanic の場合の Name  $\Rightarrow$  Title 変換の様に、( Name のままでは) 決定木がモデルに組み込みにくい場合には手助け(変数の作成)が必要
  - 決定木は、分岐のくり返しによって変数間の交互作用を反映することができる一方、変数間の(和、)差、比などを表現することが難しいので、計算することに意味のある範囲内で変数間の(和、)差、比を計算して新たな変数を作成することは有益



## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回: 2 値データの分類問題 [Titanic]
- 第 6 回: 回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

# データの読み込み、前処理

```

import optuna
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import KFold
from sklearn.metrics import log_loss

# train.csv と test.csv の結合
df1 = pd.read_csv('C:/py/titanic/train.csv', header=0)
df0 = pd.read_csv('C:/py/titanic/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0
df = pd.concat([df1, df0])

# Name について新たな変数 Title を作成（した後削除）
df['Title'] = df['Name'].apply(lambda x:
x.split(',') [1]).apply(lambda x: x.split() [0])

# カテゴリ変数の数値化 (one-hot encoding)
cat_vars = ['Pclass', 'Sex', 'Ticket', 'Cabin', 'Embarked', 'Title']
for c in cat_vars:
    df = pd.concat([df.drop(labels=[c], axis=1), pd.get_dummies(df[c],
        dummy_na=True, drop_first=False, prefix=c)], axis=1)

# 学習用データとテストデータに分割
drop_vars = ['PassengerId', 'Survived', 'Is_train', 'Name']
train_x = df.query('Is_train == 1').drop(drop_vars, axis=1)
train_y = df.query('Is_train == 1')['Survived']
test_x = df.query('Is_train == 0').drop(drop_vars, axis=1)
id = df.query('Is_train == 0')['PassengerId']

```

# optuna でパラメータ・チューニング

```

def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'booster': 'gbtree',
              'objective': 'binary:logistic',
              'eval_metric': 'logloss',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.1),
              'max_depth': trial.suggest_int('max_depth', 1, 9),
              'min_child_weight': trial.suggest_loguniform('min_child_weight', 1e-20, 10),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=10000,
                            early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                            verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-logloss-mean'].values[-1]
    return best_score

```

※ `trial.suggest_loguniform()` は `np.log(0.1)～np.log(10)` の 0.1 や 10 を指定する

# optuna でパラメータ・チューニング

```
SEED      = 777
N_FOLDS  = 4
study = optuna.create_study(direction='minimize')    # 又は 'maximize'
study.optimize(objective, n_trials=100, timeout=5000)
trial = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

# チューニング後のパラメータで予測

```
N_ESTIMATORS = 2633
params = {'booster': 'gbtree',
          'objective': 'binary:logistic',
          'eta': 0.01912092320816656,
          'max_depth': 5,
          'min_child_weight': 1.6690550323680652e-15,
          'colsample_bytree': 0.11740936148113129,
          'colsample_bylevel': 0.24537198862427906,
          'colsample_bynode': 0.4714828392699139,
          'subsample': 0.4573073142360151,
          'gamma': 4.396719621771756e-19,
          'alpha': 1.6815424853472815e-18,
          'lambda': 5.002784484922577e-15,
          'grow_policy': 'lossguide',
          'random_state': 123
        }
```

kaggle のコンペの  
score: 0.79665  
863位/18624人中

# ハイパーパラメータの設定、学習の実行、予測、提出用データ

```
num_round    = N_ESTIMATORS
dtrain       = xgb.DMatrix(train_x, label=train_y)
dtest        = xgb.DMatrix(test_x)
model        = xgb.train(params, dtrain, num_round)
pred         = model.predict(dtest)
pred_label   = np.where(pred > 0.5, 1, 0)
out          = pd.DataFrame({'PassengerId':id, 'Survived':pred_label})
out.to_csv('C:/py/titanic/gender_submission.csv', index=False)
```

※ アンサンブル（同じ作業を 5 回くり返して予測値の平均を算出）したが、改善せず

## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回：2 値データの分類問題 [Titanic]
- 第 6 回：回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

# データの読み込み、前処理

```

import optuna
import warnings
import numpy          as np
import pandas         as pd
import matplotlib.pyplot as plt
import seaborn        as sns
import xgboost        as xgb
from   sklearn.model_selection import KFold
from   sklearn.metrics      import mean_squared_error
warnings.simplefilter('ignore', FutureWarning) # 警告を非表示

# train.csv と test.csv の結合
df1 = pd.read_csv('C:/py/housing/train.csv', header=0)
df0 = pd.read_csv('C:/py/housing/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0
df  = pd.concat([df1, df0])

# データの間違い修正
all_vars = df1.columns.to_list()
cat_vars = df1.select_dtypes(include='object').columns.to_list() +
['MSSubClass', 'MoSold', 'YrSold']
num_vars = [x for x in all_vars if x not in cat_vars +
['Id', 'SalePrice', 'Is_train']]

x = 'GarageYrBlt'
df[x] = df[x].apply(lambda x : 2007 if x == 2207 else x)

```

# 前処理

```
# 外れ値処理
for x in num_vars:
    lower = np.mean(df[x]) - 3 * np.std(df[x])
    upper = np.mean(df[x]) + 3 * np.std(df[x])
    df[x] = np.where(df[x] < lower, lower, df[x])
    df[x] = np.where(df[x] > upper, upper, df[x])

# カテゴリ変数の数値化 (one-hot encoding)
for c in cat_vars:
    df = pd.concat([df.drop(labels=[c], axis=1),
                    pd.get_dummies(df[c], dummy_na=True, drop_first=False,
                                   prefix=c)], axis=1)

# データの分割
train_x = df.query('Is_train == 1').drop(['Id', 'SalePrice',
                                             'Is_train'], axis=1)
train_y = np.log1p(df.query('Is_train == 1')['SalePrice'])
test_x = df.query('Is_train == 0').drop(['Id', 'SalePrice',
                                         'Is_train'], axis=1)
id      = df.query('Is_train == 0')['Id']
```

入れた方が良いかも...

# *optuna* でパラメータ・チューニング

```

def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'booster': 'gbtree',
              'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.1),
              'max_depth': trial.suggest_int('max_depth', 1, 10),
              'min_child_weight': trial.suggest_loguniform('min_child_weight', 1e-20, 10),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=5000,
                           early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                           verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-rmse-mean'].values[-1]
    return best_score

```

# optuna でパラメータ・チューニング

```
SEED      = 777
N_FOLDS  = 4
study     = optuna.create_study(direction='minimize')      # 'maximize'
study.optimize(objective, n_trials=100, timeout=10000)
trial     = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

# チューニング後のパラメータで予測

```
N_ESTIMATORS = 4890
params = {'booster': 'gbtree',
'objective': 'reg:squarederror',
'eta': 0.005107858908628096,
'max_depth': 5,
'min_child_weight': 4.364977408921935e-11,
'colsample_bytree': 0.3624543386127742,
'colsample_bylevel': 0.5793388349683782,
'colsample_bynode': 0.620049058938708,
'subsample': 0.5458169704089749,
'gamma': 5.076210671136954e-20,
'alpha': 0.015570353464057784,
'lambd': 9.549319826840685e-20,
'grow_policy': 'depthwise',
'random_state': 2345678
}

# xgboost 用のデータ構造に変換
dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)

# ハイパーパラメータの設定、学習の実行、予測、提出用データ
num_round = N_ESTIMATORS
model      = xgb.train(params, dtrain, num_round)
pred       = np.expm1(model.predict(dtest))
out        = pd.DataFrame({'Id':id, 'SalePrice':pred})
out.to_csv('C:/py/housing/submission.csv', index=False)
```

kaggle のコンペの  
score: 0.12119  
456位/4734人中

※ アンサンブル（同じ作業を 9 回くり返して予測値の平均を算出）したが、改善せず

## 参考: パラメータ・チューニング時の枝刈り

```
# optuna
def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'booster': 'gbtree',
              'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.05),
              'max_depth': trial.suggest_int('max_depth', 4, 10),
              'min_child_weight': trial.suggest_loguniform('min_child_weight', 1e-20, 10),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    pruning_callback = optuna.integration.XGBoostPruningCallback(trial, 'test-rmse')
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=5000,
                           early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                           verbose_eval=False, callbacks=[pruning_callback])
    trial.set_user_attr('n_estimators', len(xgb_cv_results)); print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-rmse-mean'].values[-1]
    return best_score
```

## 参考: パラメータ・チューニング時の枝刈り

```
# 実行
SEED      = 777
N_FOLDS  = 4

# optuna.pruners.MedianPruner(n_warmup_steps=20)
study   = optuna.create_study(pruner=optuna.pruners.PercentilePruner(75,
  n_warmup_steps=10), direction='minimize')

study.optimize(objective, n_trials=100, timeout=10000)
trial  = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

※ [https://github.com/optuna/optuna/blob/master/examples/xgboost/xgboost\\_cv.py](https://github.com/optuna/optuna/blob/master/examples/xgboost/xgboost_cv.py)

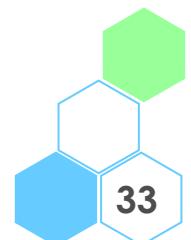
## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回：2 値データの分類問題 [Titanic]
- 第 6 回：回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

## おまけ: *Tabular Playground Series – Jan 2021*

- 18 頁の方針が悪くないかどうか検証するため別のコンペに参加
- [kaggle](#) のコンペ: [Tabular Playground Series – Jan 2021](#)
  - Practice your ML regression skills on this approachable dataset! とのこと
  - 14 個の連続変数(cont1 ~ cont14)より、連続変数(target)の値を予測する
  - 指標は Root MSE(RMSE)
- train.csv(学習用データ:30 万)と test.csv(テストデータ:20 万)がある
  - 欠測なし
  - 連続変数(cont1 ~ cont14)と目的変数(target)との相関は、ほぼ無し
  - 目的変数(target)の外れ値処理のみ行い、後はそのまま使用
  - データが巨大すぎるため扱うのが難しい
    - Google Colab. の GPU/TPU を使用
    - XGBoost の tree\_method を 'gpu\_hist' 又は 'hist' に変更
    - XGBoost の max\_depth, min\_child\_weight につき、大きめの値も探索する
- [Get Started: Jan Tabular Playground Competition](#) より
  - Dummy Regressor(全予測値を中心値にする): RMSE = 0.73487
  - Simple Linear Regression: RMSE = 1.33039
  - Lasso Regression: RMSE = 2.38477
  - Random Forest: RMSE = 0.71410



# データの読み込み、前処理

```

import optuna
import warnings
import numpy           as np
import pandas          as pd
import matplotlib.pyplot as plt
import seaborn         as sns
import xgboost          as xgb
from   sklearn.model_selection import KFold
from   sklearn.metrics      import mean_squared_error
warnings.simplefilter('ignore', FutureWarning) # 警告を非表示

# train.csv と test.csv の結合
from google.colab import drive
drive.mount('/content/drive')

df1 = pd.read_csv('drive/My Drive/ML/train.csv', header=0)
df0 = pd.read_csv('drive/My Drive/ML/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0

# データの結合
df  = pd.concat([df1, df0])

# 目的変数 (target) の外れ値処理
x = 'target'
lower = np.mean(df[x]) - 3 * np.std(df[x])
upper = np.mean(df[x]) + 3 * np.std(df[x])
df[x] = np.where(df[x] < lower, lower, df[x])
df[x] = np.where(df[x] > upper, upper, df[x])

# データの分割
train_x = df.query('Is_train == 1').drop(['id', 'target', 'Is_train'], axis=1)
train_y = df.query('Is_train == 1')['target']
test_x  = df.query('Is_train == 0').drop(['id', 'target', 'Is_train'], axis=1)
id      = df.query('Is_train == 0')['id']

```

# optuna でパラメータ・チューニング

```

def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'tree_method':'gpu_hist',
              'booster': 'gbtree',
              'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.15),
              'max_depth': trial.suggest_int('max_depth', 4, 20, step=2),
              'min_child_weight': trial.suggest_int('min_child_weight', 1, 296, step=5),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=10000,
                           early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                           verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-rmse-mean'].values[-1]
    return best_score

```

# optuna でパラメータ・チューニング

```
SEED      = 777
N_FOLDS  = 4
study     = optuna.create_study(direction='minimize')      # 'maximize'
study.optimize(objective, n_trials=100, timeout=10000)
trial     = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

# チューニング後のパラメータで予測

```

num_round = 5000
params = {'booster': 'gbtree',
          'tree_method':'hist',    # or 'gpu_hist'
          'objective': 'reg:squarederror',
          'eta': 0.0026652345119941093,
          'max_depth': 20,
          'min_child_weight': 201,
          'colsample_bytree': 0.701455902058216,
          'colsample_bylevel': 0.6527063415503433,
          'colsample_bynode': 0.9526442923547813,
          'subsample': 0.7121990965820921,
          'gamma': 1.5827649746336096e-19,
          'alpha': 6.739455100401646e-05,
          'lambda': 3.2153536566797412e-12,
          'grow_policy': 'lossguide',
          'random_state': 7777777
        }

# xgboost 用のデータ構造に変換
dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)

# ハイパーパラメータの設定、学習の実行、予測、提出用データ
model      = xgb.train(params, dtrain, num_round)
pred       = np.expm1(model.predict(dtest))
out        = pd.DataFrame({'Id':id, 'SalePrice':pred})
out.to_csv('C:/py/housing/submission.csv', index=False)

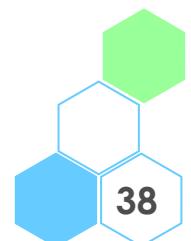
```

kaggle のコンペの  
score: 0.69620  
304位/1728人中

※ アンサンブル（同じ作業を 8 回くり返して予測値の平均を算出）したが、あまり改善せず（0.69618、301位）

## おまけ: *Tabular Playground Series – Feb 2021*

- 18 頁の方針が悪くないかどうか検証するため別のコンペに参加
- [kaggle](#) のコンペ: [Tabular Playground Series – Feb 2021](#)
  - Practice your ML regression skills on this approachable dataset! とのこと
  - 10 個のカテゴリ変数(cat0 ~ cat9)と 14 個の連続変数(cont0 ~ cont13)より、連続変数(target)の値を予測する
  - 指標は Root MSE(RMSE)
- train.csv(学習用データ:30 万)と test.csv(テストデータ:20 万)がある
  - 欠測なし
  - 連続変数(cont0 ~ cont13)と目的変数(target)との相関は、ほぼ無し
  - データが巨大すぎるため扱うのが難しい
    - Google Colab. の GPU/TPU を使用
    - XGBoost の tree\_method を 'gpu\_hist' 又は 'hist' に変更
    - XGBoost の max\_depth、min\_child\_weight につき、大きめの値も探索する
- [Get Started: FEB Tabular Playground Competition](#) より
  - Dummy Regressor(全予測値を中心値にする): RMSE = 0.88857
  - Simple Linear Regression: RMSE = 1.00779
  - Lasso Regression: RMSE = 1.33854
  - Random Forest: RMSE = 0.86391



# データの読み込み、前処理

```
import optuna
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
warnings.simplefilter('ignore', FutureWarning) # 警告を非表示

# train.csv と test.csv の結合
from google.colab import drive
drive.mount('/content/drive')

df1 = pd.read_csv('drive/My Drive/ML/train.csv', header=0)
df0 = pd.read_csv('drive/My Drive/ML/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0

# データの結合
df = pd.concat([df1, df0])
```

# データの読み込み、前処理

```

# カテゴリ変数の数値化
# df.select_dtypes(include='object').columns.to_list()
cat_vars = ['cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6',
            'cat7', 'cat8', 'cat9']
for c in cat_vars:
    df = pd.concat([df.drop(labels=[c], axis=1), pd.get_dummies(df[c],
                                                               dummy_na=True, drop_first=False, prefix=c)], axis=1)

# 目的変数(target) の外れ値処理
x = 'target'
lower = np.mean(df[x]) - 3 * np.std(df[x])
upper = np.mean(df[x]) + 3 * np.std(df[x])
df[x] = np.where(df[x] < lower, lower, df[x])
df[x] = np.where(df[x] > upper, upper, df[x])

# データの分割
train_x = df.query('Is_train == 1').drop(['id', 'target', 'Is_train'],
axis=1)
train_y = df.query('Is_train == 1')['target']
test_x = df.query('Is_train == 0').drop(['id', 'target', 'Is_train'],
axis=1)
id      = df.query('Is_train == 0')['id']

```

# *optuna* でパラメータ・チューニング

```

def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'booster': 'gbtree',
              'objective': 'reg:squarederror',
              'eval_metric': 'rmse',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.15),
              'max_depth': trial.suggest_int('max_depth', 4, 20, step=2),
              'min_child_weight': trial.suggest_int('min_child_weight', 1, 296, step=5),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise',
                                                               'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=10000,
                           early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                           verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-rmse-mean'].values[-1]
    return best_score

```

# optuna でパラメータ・チューニング

```
SEED      = 777
N_FOLDS  = 5
study     = optuna.create_study(direction='minimize')      # 'maximize'
study.optimize(objective, n_trials=10, timeout=10000)
trial     = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

# チューニング後のパラメータで予測

```

num_round = 3405
params = {'booster': 'gbtree',
'objective': 'reg:squarederror',
'eta': 0.020476295251566204,
'max_depth': 4,
'min_child_weight': 206,
'colsample_bytree': 0.5695052358709458,
'colsample_bylevel': 0.7425994452724571,
'colsample_bynode': 0.6763015944358355,
'subsample': 0.3870856402596746,
'gamma': 2.1811733935400702e-06,
'alpha': 5.451706054156088e-07,
'lambd': 0.3629656233255221,
'random_state': 4444
}

```

# xgboost 用のデータ構造に変換

```

dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)

```

# ハイパーパラメータの設定、学習の実行、予測、提出用データ

```

model      = xgb.train(params, dtrain, num_round)
pred       = np.expm1(model.predict(dtest))
out        = pd.DataFrame({'Id':id, 'SalePrice':pred})
out.to_csv('C:/py/housing/submission.csv', index=False)

```

kaggle のコンペの  
score: 0.84266

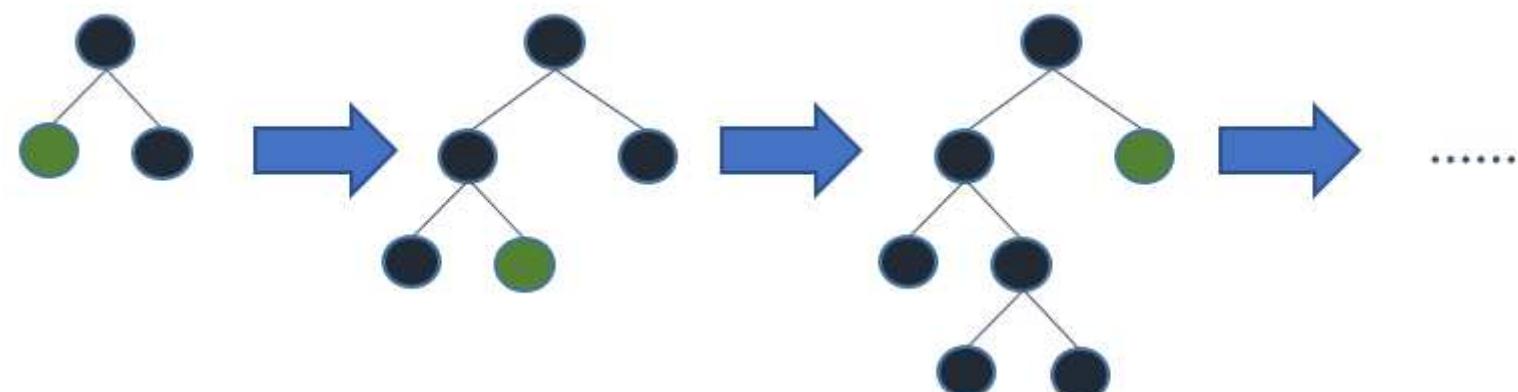
※ アンサンブル（同じ作業を 8 回くり返して予測値を平均）し、スコアは 0.84238 に改善 (323位/1433人中)

☆ ハイスペックな環境であれば optuna がクルクル回るのだろうが、半日実行して10~15回が限度・・・。

そこでお試しで LightGBM を試してみる

## 参考: *LightGBM*

- Documentation  
<https://lightgbm.readthedocs.io/en/latest/>
- 論文「 LightGBM: A Highly Efficient Gradient Boosting Decision Tree. 」  
<https://papers.nips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf>
- XGBoostと同じGBDTであるが、工夫・高速化のための改良がなされている
  - 数値データを「ヒストグラムの各棒」に分け離散データに落とし込み、「分岐時のスコア計算にこれを使用」「葉を追加する際のヒストグラムを構成する際に、上の葉と近くのヒストグラムとの引き算で計算」等の工夫により高速化・メモリ節約を実現
  - 木を成長させる際、レベル(深さ)ごとではなく葉ごとに成長するかどうかを判定
  - 他にも色々な工夫あり( label-encoded なカテゴリ変数に対する処理、GPU 対応)



※ 出典:LightGBM Features <https://lightgbm.readthedocs.io/en/latest/Features.html>

# 参考: LightGBM + optuna をお試し①

```
### 学習+パラメータチューニング
import optuna.integration.lightgbm as lgb

SEED    = 777
dtrain = lgb.Dataset(train_x, label=train_y)
dtest  = lgb.Dataset(test_x)

# objective: regression, binary, multiclass, etc., metric:
params = {'random_state': SEED, 'objective': 'regression', 'metric': 'rmse',
           'verbosity': -1, 'boosting_type': "gbdt", 'learning_rate': 0.01}
tuner  = lgb.LightGBMTunerCV(params, dtrain, num_boost_round=10000, seed=SEED,
                               verbose_eval=100, early_stopping_rounds=100, folds=KFold(n_splits=5),
                               stratified=False)
tuner.run()
print("Best score:", tuner.best_score)
best_params = tuner.best_params
print("Best params:", best_params)

### 予測
import lightgbm as lgb_

params = {'random_state': 7777777, 'objective': 'regression', 'metric': 'rmse',
           'verbosity': -1, 'boosting_type': 'gbdt', 'learning_rate': 0.01,
           'feature_pre_filter': False, 'lambda_l1': 6.463676395948396,
           'lambda_l2': 0.08647391756846819, 'num_leaves': 95, 'feature_fraction': 0.4,
           'bagging_fraction': 0.8361202898801138, 'bagging_freq': 3,
           'min_child_samples': 25}
gbm     = lgb_.train(params, dtrain, num_boost_round=2200)
pred   = gbm.predict(test_x)
out    = pd.DataFrame({'id':id, 'target':pred})
out.to_csv('drive/My Drive/ML/submission.csv', index=False)
```

kaggle のコンペの  
score: 0.84268

## 参考: LightGBM + optuna をお試し②

```

import optuna
import lightgbm as lgb

SEED      = 333
N_FOLDS  = 5
d_train = lgb.Dataset(train_x, label=train_y)

def objective(trial):
    param = {'seed': SEED,
              'bagging_seed':SEED,
              'feature_fraction_seed':SEED,
              'objective': 'regression', # binary, multiclass
              'metric': 'rmse', # mae, mse, auc, binary_logloss, binary_error, multi_logloss, etc.
              'verbosity': -1,
              'boosting_type': 'gbdt',
              'feature_pre_filter': False,
# 'num_boost_round': trial.suggest_int('num_boost_round', 50, 10000),
# 'max_depth': trial.suggest_int('max_depth', 1, 20), # -1 (no limit)
              'learning_rate': trial.suggest_float('learning_rate', 1e-20, 0.1),
              'bagging_fraction': trial.suggest_float('bagging_fraction', 1e-20, 1.0), # subsample
              'bagging_freq': trial.suggest_int('bagging_freq', 1, 10), # subsample_freq
              'feature_fraction': trial.suggest_float('feature_fraction', 1e-20, 1.0), # colsample_bytree
              'feature_fraction_bynode': trial.suggest_float('feature_fraction_bynode', 1e-20, 1.0), # colsample_bynode
              'num_leaves': trial.suggest_int('num_leaves', 2, 256), # max_leaves
              'min_child_samples': trial.suggest_int('min_child_samples', 1, 300), # min_data_in_leaf
              'min_data_per_group': trial.suggest_int('min_data_per_group', 10, 300),
              'max_bin': trial.suggest_int('max_bin', 128, 1024),
              'lambda_l1': trial.suggest_float('lambda_l1', 1e-20, 20.0, log=True),
              'lambda_l2': trial.suggest_float('lambda_l2', 1e-20, 20.0, log=True)
    }

    cv_results = lgb.cv(param, d_train, num_boost_round=10000, early_stopping_rounds=100,
                        nfold=N_FOLDS, stratified=False, seed=SEED, verbose_eval=False) # return_cvbooster=True
    trial.set_user_attr('n_estimators', len(cv_results['rmse-mean']))
    print(len(cv_results['rmse-mean']))
    best_score = cv_results['rmse-mean'] [-1]
    return best_score
}

```

## 参考: LightGBM + optuna をお試し②

```
study = optuna.create_study(direction='minimize')      # 'maximize'
study.optimize(objective, n_trials=50, timeout=36000)
trial = study.best_trial
N_ESTIMATORS = trial.user_attrs['n_estimators']

print('Number of finished trials: {}'.format(len(study.trials)))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Number of estimators: {}'.format(N_ESTIMATORS))
```

### ### 学習

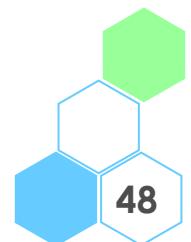
```
SEED    = 33333
params = {'seed': SEED, 'bagging_seed':SEED, 'feature_fraction_seed':SEED,
          'objective': 'regression', 'metric': 'rmse', 'verbosity': -1,
          'boosting_type': 'gbdt', 'feature_pre_filter': False,
          'learning_rate': 0.006233654908617756, 'bagging_fraction': 0.7431509976076313,
          'bagging_freq': 10, 'feature_fraction': 0.2402196233595903,
          'feature_fraction_bynode': 0.9278627226248137, 'num_leaves': 64,
          'min_child_samples': 174, 'min_data_per_group': 272,
          'max_bin': 350, 'lambda_l1': 1.4110723926048064e-16,
          'lambda_l2': 1.3910698990853774e-08}
gbm     = lgb.train(params, dtrain, num_boost_round=4150)
```

kaggle のコンペの  
score: 0.84227

※ アンサンブル（同じ作業を 6 回くり返して予測値の平均を算出）し、スコアは 0.84205 に改善、  
Private Score=0.84267 (205位/1433人中) , Public Score=0.84205 (226位/1433人中)

## 参考: *Tabular Playground Series – Mar 2021*

- LightGBM の練習(分類タスクの練習)のため別のコンペに参加
  - 動くかどうか試すだけでスコアアップは目指さない
- [kaggle](#) のコンペ: [Tabular Playground Series – Mar 2021](#)
  - Practice your ML regression skills on this approachable dataset! とのこと
  - 19 個のカテゴリ変数(cat0 ~ cat18)と 11 個の連續変数(cont0 ~ cont10)より、0-1 の 2 値変数(target)の値を予測する
  - 指標は [ROC 曲線の AUC](#)
- train.csv(学習用データ:30 万)と test.csv(テストデータ:20 万)がある
  - 欠測なし
  - 連續変数(cont0 ~ cont13)と目的変数(target)との相関は、ほぼ無し
  - データが巨大すぎるため扱うのが難しい
    - Google Colab. の GPU を使用



# 参考: データの読み込み、前処理

```

import optuna
import warnings
import numpy           as np
import pandas          as pd
import matplotlib.pyplot as plt
import seaborn         as sns
import lightgbm        as lgb
from   sklearn.model_selection import KFold
from   sklearn.metrics      import mean_squared_error
warnings.simplefilter('ignore', FutureWarning) # 警告を非表示

# train.csv と test.csv の結合
from google.colab import drive
drive.mount('/content/drive')
df1 = pd.read_csv('drive/My Drive/ML/train.csv', header=0)
df0 = pd.read_csv('drive/My Drive/ML/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0

# データの結合
df  = pd.concat([df1, df0])

# カテゴリ変数の数値化
cat_vars = ['cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7',
            'cat8', 'cat9', 'cat10', 'cat11', 'cat12', 'cat13', 'cat14', 'cat15', 'cat16',
            'cat17', 'cat18']
for c in cat_vars:
    df = pd.concat([df.drop(labels=[c], axis=1), pd.get_dummies(df[c],
                                                               dummy_na=True, drop_first=False, prefix=c)], axis=1)

# データの分割
train_x = df.query('Is_train == 1').drop(['id', 'target', 'Is_train'], axis=1)
train_y = df.query('Is_train == 1')['target']
test_x  = df.query('Is_train == 0').drop(['id', 'target', 'Is_train'], axis=1)
id      = df.query('Is_train == 0')['id']

```

## 参考: *optuna* でパラメータチューニング

```

SEED      = 777
N_FOLDS  = 5
d_train  = lgb.Dataset(train_x, label=train_y)

def objective(trial):
    param = {'seed': SEED,
              'bagging_seed':SEED,
              'feature_fraction_seed':SEED,
              'objective': 'binary', # regression, multiclass
              'metric': 'auc', # mae, mse, auc, binary_logloss, binary_error, multi_logloss, multi_error
              'verbosity': -1,
              'boosting_type': 'gbdt',
              'feature_pre_filter': False,
              'learning_rate': trial.suggest_float('learning_rate', 1e-20, 0.1),
              'bagging_fraction': trial.suggest_float('bagging_fraction', 1e-20, 1.0),
              'bagging_freq': trial.suggest_int('bagging_freq', 1, 10),
              'feature_fraction': trial.suggest_float('feature_fraction', 1e-20, 1.0),
              'feature_fraction_bynode': trial.suggest_float('feature_fraction_bynode', 1e-20, 1.0),
              'num_leaves': trial.suggest_int('num_leaves', 2, 256),
              'min_child_samples': trial.suggest_int('min_child_samples', 1, 300),
              'min_data_per_group': trial.suggest_int('min_data_per_group', 10, 300),
              'max_bin': trial.suggest_int('max_bin', 128, 1024),
              'lambda_l1': trial.suggest_float('lambda_l1', 1e-20, 20.0, log=True),
              'lambda_l2': trial.suggest_float('lambda_l2', 1e-20, 20.0, log=True)
    }
    cv_results = lgb.cv(param, d_train, num_boost_round=10000, early_stopping_rounds=100,
                        nfold=N_FOLDS, stratified=False, seed=SEED, verbose_eval=False)
    trial.set_user_attr('n_estimators', len(cv_results['auc-mean']))
    print(len(cv_results['auc-mean']))
    best_score = cv_results['auc-mean'][-1]
    return best_score

study = optuna.create_study(direction='maximize') # 'minimize'
study.optimize(objective, n_trials=100, timeout=50000)
trial = study.best_trial
N_ESTIMATORS = trial.user_attrs['n_estimators']
print('Number of finished trials: {}'.format(len(study.trials)))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Number of estimators: {}'.format(N_ESTIMATORS))

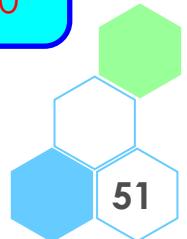
```

## 参考: チューニング後のパラメータで予測

```
SEED      = 777
params = {'seed': SEED, 'bagging_seed':SEED,
          'feature_fraction_seed':SEED, 'objective': 'binary',
          'metric': 'auc', 'verbosity': -1, 'boosting_type': 'gbdt',
          'feature_pre_filter': False, 'learning_rate': 0.00582914867602773,
          'bagging_fraction': 0.7368935708961051, 'bagging_freq': 9,
          'feature_fraction': 0.2770346515244113,
          'feature_fraction_bynode': 0.2961752862107485,
          'num_leaves': 210, 'min_child_samples': 93, 'min_data_per_group': 234,
          'max_bin': 432, 'lambda_l1': 0.0002798107152997289,
          'lambda_l2': 8.894088216212114e-12}

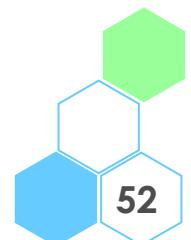
gbm      = lgb.train(params, d_train, num_boost_round=3998)
pred    = gbm.predict(test_x)
out     = pd.DataFrame({'id':id, 'target':pred})
out.to_csv('drive/My Drive/ML/submission.csv', index=False)
```

kaggle のコンペの  
score: 0.89090



## 参考: *Tabular Playground Series – Apr 2021*

- XGBoost & LightGBM の練習(分類タスクの練習)のため別のコンペに参加
  - 動くかどうか試すだけでスコアアップは目指さない
- [kaggle](#) のコンペ: [Tabular Playground Series – Apr 2021](#)
  - データセットは Titanic データからの合成で、実際の Titanic データと統計的に似せているもの
  - 変数は第 5 回の Titanic データと同じく、PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked
    - ただし中身は結構異なる(例:Name)
  - 指標は [Accuracy](#)
- train.csv(学習用データ:10 万)と test.csv(テストデータ:10 万)がある
  - データは少し大きいが、Playground Series Jan – Mar に比べると格段にマシ
    - …だが、全てのカテゴリ変数を One-hot Encoding で数値に変換するとメモリが尽きるため、Cabin 等のカテゴリが多い変数は Label Encoding (LightGBM では cat\_smooth や cat\_l2 を調整、効いているかは微妙)
    - Google Colab. の TPU を使用



# 参考：データの読み込み、前処理

```

import optuna
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import log_loss

# train.csv と test.csv の結合
from google.colab import drive
drive.mount('/content/drive')
df1 = pd.read_csv('drive/My Drive/ML/train.csv', header=0)
df0 = pd.read_csv('drive/My Drive/ML/test.csv', header=0)
df1["Is_train"] = 1
df0["Is_train"] = 0
df = pd.concat([df1, df0])

# カテゴリ変数の数値化①
cat_vars = ['Pclass', 'Sex', 'Embarked']
for c in cat_vars:
    df = pd.concat([df.drop(labels=[c], axis=1), pd.get_dummies(df[c], dummy_na=True, drop_first=False, prefix=c)], axis=1)

# カテゴリ変数の数値化②
# Name
df['LastName'] = df['Name'].apply(lambda x: x.split(',') [0]).apply(lambda x: x.split() [0])
df['FirstName'] = df['Name'].apply(lambda x: x.split(',') [1]).apply(lambda x: x.split() [0])

# 欠測かどうかの変数
df['Ticket_NA'] = df['Ticket'].isnull().astype('int')
df['Cabin_NA'] = df['Cabin'].isnull().astype('int')

cat_vars = ['Ticket', 'Cabin', 'LastName', 'FirstName']
for c in cat_vars:
    df[c].fillna("Missing", inplace=True)
for c in cat_vars:
    le = LabelEncoder()
    df[c] = le.fit_transform(df[c])

# 学習用データとテストデータに分割
drop_vars = ['PassengerId', 'Survived', 'Is_train', 'Name']
train_x = df.query('Is_train == 1').drop(drop_vars, axis=1)
train_y = df.query('Is_train == 1')['Survived']
test_x = df.query('Is_train == 0').drop(drop_vars, axis=1)
id = df.query('Is_train == 0')['PassengerId']

```

## 参考: *optuna + XGBoost*

```

def objective(trial):
    dtrain = xgb.DMatrix(train_x, label=train_y)
    dtest = xgb.DMatrix(test_x)

    param = {'seed': SEED,
              'booster': 'gbtree',
              'objective': 'binary:logistic',
              'eval_metric': 'logloss',
              'eta': trial.suggest_uniform('eta', 1e-20, 0.1),
              'max_depth': trial.suggest_int('max_depth', 3, 9),
              'min_child_weight': trial.suggest_int('min_child_weight', 1, 296, step=5),
              'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
              'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
              'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
              'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
              'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
              'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
              'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=10000, early_stopping_rounds=100,
                           nfold=N_FOLDS, seed=SEED, stratified=False, verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-logloss-mean'].values[-1]
    return best_score

SEED = 777
N_FOLDS = 5
study = optuna.create_study(direction='minimize') # 'maximize'
study.optimize(objective, n_trials=100, timeout=36000)
trial = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params

```

## 参考: *optuna + XGBoost*

```

num_round = 391
params = {'booster': 'gbtree',
'objective': 'binary:logistic',
'eta': 0.027127323092044606,
'max_depth': 9,
'min_child_weight': 86,
'colsample_bytree': 0.41140887045137753,
'colsample_bylevel': 0.9881209917138358,
'colsample_bynode': 0.8780728765082891,
'subsample': 0.9523572180045136,
'gamma': 1.0877215241146765e-14,
'alpha': 7.385828431577176e-14,
'lambd': 5.289123394782853e-10,
'grow_policy': 'depthwise',
'random_state': 777
}
# xgboost 用のデータ構造に変換
dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)

# ハイパーパラメータの設定、学習の実行、予測、提出用データ
model      = xgb.train(params, dtrain, num_round)
pred       = model.predict(dtest)
pred_label = np.where(pred > 0.5, 1, 0)
out        = pd.DataFrame({'PassengerId':id, 'Survived':pred_label})
out.to_csv('drive/My Drive/ML/submission.csv', index=False)

```

kaggle のコンペの  
 score: 0.79450

# 参考: *optuna + LightGBM*

```

import lightgbm as lgb
d_train = lgb.Dataset(train_x, label=train_y)
SEED    = 777
N_FOLDS = 4

def objective(trial):
    param = {'seed': SEED,
              'bagging_seed':SEED,
              'feature_fraction_seed':SEED,
              'objective': 'binary', # regression, multiclass
              'metric': 'binary_logloss', # mae, mse, auc, binary_logloss, binary_error, multi_logloss, multi_error
              'verbosity': -1,
              'boosting_type': 'gbdt',
              'feature_pre_filter': False,
#             'num_boost_round': trial.suggest_int('num_boost_round', 50, 10000),
#             'max_depth': trial.suggest_int('max_depth', 3, 50), # -1 (no limit)
              'learning_rate': trial.suggest_float('learning_rate', 1e-20, 0.1),
              'bagging_fraction': trial.suggest_float('bagging_fraction', 1e-20, 1.0), # subsample
              'bagging_freq': trial.suggest_int('bagging_freq', 1, 10), # subsample_freq
              'feature_fraction': trial.suggest_float('feature_fraction', 1e-20, 1.0), # colsample_bytree
              'feature_fraction_bynode': trial.suggest_float('feature_fraction_bynode', 1e-20, 1.0), # colsample_bynode
              'num_leaves': trial.suggest_int('num_leaves', 2, 256), # max_leaves
              'min_child_samples': trial.suggest_int('min_child_samples', 1, 300), # min_data_in_leaf
              'min_data_per_group': trial.suggest_int('min_data_per_group', 10, 300),
              'max_bin': trial.suggest_int('max_bin', 128, 1024),
              'lambda_11': trial.suggest_float('lambda_11', 1e-20, 20.0, log=True),
              'lambda_12': trial.suggest_float('lambda_12', 1e-20, 20.0, log=True),
              'cat_smooth' : trial.suggest_float('cat_smooth', 1e-20, 256),
              'cat_12' : trial.suggest_float('cat_12', 1e-20, 20)
    }
    cv_results = lgb.cv(param, d_train, num_boost_round=10000, early_stopping_rounds=100, nfold=N_FOLDS,
                         stratified=False, seed=SEED, verbose_eval=False) # return_cvbooster=True
    trial.set_user_attr('n_estimators', len(cv_results['binary_logloss-mean']))
    print(len(cv_results['binary_logloss-mean']))
    best_score = cv_results['binary_logloss-mean'][-1]
    return best_score

study = optuna.create_study(direction='minimize') # 'maximize'
study.optimize(objective, n_trials=200, timeout=36000)
trial = study.best_trial
N_ESTIMATORS = trial.user_attrs['n_estimators']

print('Number of finished trials: {}'.format(len(study.trials)))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Number of estimators: {}'.format(N_ESTIMATORS))

```

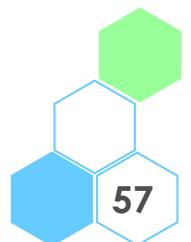
## 参考: *optuna + LightGBM*

```

SEED      = 777
params     = {'seed': SEED, 'bagging_seed':SEED, 'feature_fraction_seed':SEED,
'objective': 'binary', 'metric': 'binary_logloss', 'verbosity': -1,
'boosting_type': 'gbdt', 'feature_pre_filter': False,
'learning_rate': 0.01008220253770751, 'bagging_fraction': 0.8851613954822289,
'bagging_freq': 5, 'feature_fraction': 0.4572333085123362,
'feature_fraction_bynode': 0.3434980975807741, 'num_leaves': 40,
'min_child_samples': 112, 'min_data_per_group': 249, 'max_bin': 385,
'lambda_11': 2.3876675316104057e-16, 'lambda_12': 3.1734720445336473,
'cat_smooth': 116.42322427951875, 'cat_12': 4.762007542176315}
gbm       = lgb.train(params, d_train, num_boost_round=966)
pred      = gbm.predict(test_x)
pred_label = np.where(pred > 0.5, 1, 0)
out       = pd.DataFrame({'PassengerId':id, 'Survived':pred_label})
out.to_csv('drive/My Drive/ML/submission.csv', index=False)

```

kaggle のコンペの  
score: 0.79530



# 参考：6位！

Overview	Data	Code	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
#	Team Name	Notebook		Team Members	Score	Entries	Last	
1	Muhammad Waqar Gul				0.80717	3	2h	
2	AMontgomerie				0.80548	1	3h	
3	Mart Jürisoo				0.79914	5	2h	
4	MLJAR AutoML	</> MLJAR AutoML - T...			0.79829	2	1h	
5	onlyWinbaseline				0.79765	2	9h	
6	NobuoFunao				0.79510	3	now	
<b>Your Best Entry ↑</b> <div style="background-color: #0072BD; color: white; padding: 5px; margin-top: 10px;">           Your submission scored 0.79510, which is an improvement of your previous score of 0.79268. Great job!           <a href="#" style="color: white; margin-left: 10px;">Tweet this!</a> </div>								
7	SHOWMAKER				0.79478	5	2h	
8	Dianwen (David) Zhao	v/s TPS Anell-3 Model			0.79389	3	20m	

58

※コンペ開始数時間後で 55 人しか参加していないため・・・

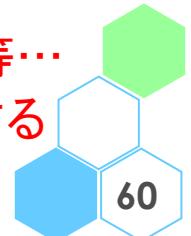
## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回：2 値データの分類問題 [Titanic]
- 第 6 回：回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

## おまけ: COMS4771 MSD Regression

- 18 頁の方針が悪くないかどうか検証するため別のコンペに参加
- [kaggle](#) のコンペ: [COMS4771 MSD Regression Competition](#)
  - データ「MSdata.mat」は音楽の曲のドメインから取得されており、20世紀から21世紀の約50万曲で構成されている
  - タスクは、曲の音楽コンテンツを考慮して、曲がリリースされた年を予測すること
  - 前処理の負担を軽減するために、各曲(Observation)はすでに 90 の高品質の音色ベースの特徴量にベクトル化されている
  - 指標は Mean Absolute Error (MAE)、[終了したコンペだが submission 可能](#)
- train.csv(学習用データ:30 万)と test.csv(テストデータ:20 万)がある
  - MSdata.mat には trainx、trainy、testx の Matlab 形式の 3 変数が含まれている:
    - trainx:  $463715 \times 90$  matrix of training data, where each row is a 90-dimensional feature representation of a song.
    - trainy:  $463715 \times 1$  vector of labels associated with each training data. Each row is the release year of the corresponding song in trainx.
    - testx:  $51630 \times 90$  matrix of test data.
  - 先ほどのコンペよりも、データがさらに巨大すぎるため扱うのが難しい
    - Google Colab. の TPU を使用
    - XGBoost の tree\_method を変更、CV の回数を 2、木の数を減らす、等…
    - XGBoost の max\_depth、min\_child\_weight につき、大きめの値も探索する  
⇒ 究極でも optuna で 5 ~ 10 回/日 が限界…



# データの読み込み、前処理

```
import optuna
import warnings
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
warnings.simplefilter('ignore', FutureWarning) # 警告を非表示

# 元データは .mat 形式
from google.colab import drive
drive.mount('/content/drive')
dic = scipy.io.loadmat('drive/My Drive/ML/MSdata.mat')

# train_x: 463715 x 90
vars = []
for i in range(90):
    vars.append('var' + str(i))
train_x = pd.DataFrame(dic['trainx'])
train_x.columns = vars

# train_y: 463715 x 1
train_y = pd.DataFrame(dic['trainy'])
train_y.columns = ['prediction']

# test_x: 51630 x 90
vars = []
for i in range(90):
    vars.append('var' + str(i))
test_x = pd.DataFrame(dic['testx'])
test_x.columns = vars

# dataid: 1, ..., 51630
dataid = range(1, 51631)
```

# *optuna* でパラメータ・チューニング

```

dtrain = xgb.DMatrix(train_x_tmp, label=train_y)
dtest = xgb.DMatrix(test_x_tmp)

def objective(trial):
    param = {'seed': SEED,
# 'tree_method':'hist',
        'booster': 'gbtree',
        'objective': 'reg:squarederror',
        'eval_metric': 'rmse',
        'eta': trial.suggest_uniform('eta', 1e-20, 0.15),
        'max_depth': trial.suggest_int('max_depth', 4, 20, step=2),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 296, step=5),
        'colsample_bytree': trial.suggest_uniform('colsample_bytree', 1e-20, 1),
        'colsample_bylevel': trial.suggest_uniform('colsample_bylevel', 1e-20, 1),
        'colsample_bynode': trial.suggest_uniform('colsample_bynode', 1e-20, 1),
        'subsample': trial.suggest_uniform('subsample', 1e-20, 1),
        'gamma': trial.suggest_loguniform('gamma', 1e-20, 1.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-20, 1.0),
        'lambda': trial.suggest_loguniform('lambda', 1e-20, 1.0),
        'grow_policy': trial.suggest_categorical('grow_policy', ['depthwise', 'lossguide'])}
    xgb_cv_results = xgb.cv(params=param, dtrain=dtrain, num_boost_round=3000,
                           early_stopping_rounds=100, nfold=N_FOLDS, seed=SEED, stratified=False,
                           verbose_eval=False)
    trial.set_user_attr('n_estimators', len(xgb_cv_results))
    print(len(xgb_cv_results))
    best_score = xgb_cv_results['test-rmse-mean'].values[-1]
    return best_score

```

# optuna でパラメータ・チューニング

```
SEED      = 777
N_FOLDS  = 2

train_x_tmp = train_x
test_x_tmp  = test_x

study      = optuna.create_study(direction='minimize')      # 'maximize'
study.optimize(objective, n_trials=10, timeout=36000)
trial      = study.best_trial

print('Number of finished trials: ', len(study.trials))
print('Best trial:')
print('  Value: {}'.format(trial.value))
print('  Params: ')
for key, value in trial.params.items():
    print('    {}: {}'.format(key, value))

N_ESTIMATORS = trial.user_attrs['n_estimators']
print('  Number of estimators: {}'.format(N_ESTIMATORS))

# 最適なパラメータ
study.best_params
```

# チューニング後のパラメータで予測

```
# xgboost 用のデータ構造に変換
dtrain = xgb.DMatrix(train_x, label=train_y)
dtest = xgb.DMatrix(test_x)

num_round = 4000
params = {'booster': 'gbtree',
'objective': 'reg:squarederror',
'alpha': 1.0776776138215445e-13,
'colsample_bylevel': 0.5625213949446577,
'colsample_bynode': 0.09573684898429258,
'colsample_bytree': 0.8727761114533332,
'eta': 0.022549140985011738,
'gamma': 2.9186077250310154e-16,
'grow_policy': 'depthwise',
'lambd': 7.142884752341316e-20,
'max_depth': 16,
'min_child_weight': 161,
'subsample': 0.8403318584740613,
'random_state': 777
}
```

kaggle のコンペ:17位/80人  
 Private Score=6.11657,  
 Public Score=6.16424

# ハイパーパラメータの設定、学習の実行、予測、提出用データ

```
model = xgb.train(params, dtrain, num_round)
pred = np.round(model.predict(dtest))
out = pd.DataFrame({'dataid':dataid, 'prediction':pred})
out.to_csv('drive/My Drive/ML/submission.csv', index=False)
```

※ アンサンブル（同じ作業を 6 回くり返して予測値の平均算出）したが改善せず： Private Score=6.11458,  
 Public Score=6.16017、もう少し計算環境に余裕があれば改善しそう…

※ LightGBM + optuna だと 15~20 回は回るので、さらに改善しそう

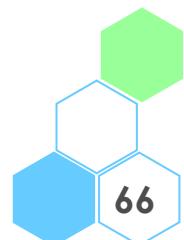
## メニュー

---

- データの概要と XGBoost の概要、予測方針
- 第 5 回：2 値データの分類問題 [Titanic]
- 第 6 回：回帰問題 [House Prices]
- おまけ
  - Tabular Playground Series – Jan 2021 – Apr 2021
  - COMS4771 MSD Regression Competition

# 参考文献

- Python 3.8.3 documentation  
<https://docs.python.org/3/>  
<https://docs.python.org/ja/3/>
- Scikit-learn documentation  
<https://scikit-learn.org/stable/>  
[https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
- XGBoost 論文 & documentation  
Tianqi Chen and Carlos Guestrin (2016) “XGBoost: A Scalable Tree Boosting System”  
<https://xgboost.readthedocs.io/en/latest/index.html>
- Sebastian Raschka & Vahid Mirjalili (2019) “Python Machine Learning, 3rd Edition”, Packt Publishing  
<https://github.com/rasbt/python-machine-learning-book-3rd-edition>
- note.nkmk.me: <https://note.nkmk.me/python/>
- kaggle: <https://www.kaggle.com/>
- 門脇 大輔 他著(2019)「Kaggleで勝つデータ分析の技術(技術評論社)」  
<https://github.com/ghmagazine/kagglebook>
- optuna: <https://optuna.readthedocs.io/en/stable/index.html>





– End of File –

