勾配ブースティング決定木の仕組み
と
データコンペ結果報告

原光太郎, 梅澤正人, 長岡理子, 山下直斗, 若松孝明, 遠藤航希, 馮雪, 相澤大揮

早稲田大学 応用数理学科

December 8, 2023

発表の目的

- nishika データコンペの結果報告.
- 価格予想を行う際に用いた, 勾配ブースティング決定木 (Gradient Boosting Decision Tree, 以下 GBDT とする) という手法とそのフレームワークである XGboost, LightGBM の紹介.

データコンペの概要

- データコンペとは
  - あらかじめ決められた課題に対し, 機械学習や統計学の知見などを用いて分析を行い, 最適解の予測精度を競うコンペティション.
- なぜ nishika のデータコンペなのか
  - 内容についても身近なテーマが多く参加しやすい.
  - Kaggle や SIGNATE は, はじめの一歩としては, 難易度が高く挫折しそう.
  - 中古マンションの価格予想が面白そう.

データコンペの概要

- 名称
  - 中古マンション価格予測 2023 夏の部
- 期間
  - 8 月から 9 月
- 利用データ
  - Nishika にて土地総合情報システムより収集・加工し作成したもの.

データコンペの概要

- なお, 使用データは以下のようになっている.

ID	種類	地域	市区町村コード	都道府県名	市区町村名	地区名	最寄駅：名称	最寄駅：距離（分）	間取り	面積（㎡）	…	前面道路：方位	前面道路：種類	前面道路：幅員（m）	都市計画	建ぺい率（%）	容積率（%）	取引時点	改装	取引の事情等	取引価格（総額）_log
40045006	中古マンション等	NaN	40136	福岡県	福岡市城南区	片江	福大前	14	2LDK	65	…	NaN	NaN	NaN	第２種住居地域	60.0	200.0	2022年第1四半期	未改装	調停・競売等	7.079181
40108482	中古マンション等	NaN	40203	福岡県	久留米市	諏訪野町	西鉄久留米	7	3LDK	75	…	NaN	NaN	NaN	商業地域	80.0	400.0	2016年第1四半期	未改装	NaN	7.255273
40162657	中古マンション等	NaN	40136	福岡県	福岡市城南区	別府	別府(福岡)	3	2LDK	60	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2008年第4四半期	未改装	NaN	7.146128
40108562	中古マンション等	NaN	40203	福岡県	久留米市	諏訪野町	花畑	13	3LDK	60	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2016年第4四半期	未改装	NaN	7.000000
40048209	中古マンション等	NaN	40137	福岡県	福岡市早良区	荒江	西新	16	2LDK	55	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2020年第3四半期	未改装	NaN	7.079181
40040759	中古マンション等	NaN	40135	福岡県	福岡市西区	拾六町	橋本(福岡)	28	3LDK	55	…	NaN	NaN	NaN	第２種住居地域	60.0	200.0	2014年第3四半期	改装済	NaN	7.079181
40054546	中古マンション等	NaN	40132	福岡県	福岡市博多区	住吉	博多	13	1K	25	…	NaN	NaN	NaN	商業地域	80.0	500.0	2015年第4四半期	未改装	NaN	7.079181
40025069	中古マンション等	NaN	40132	福岡県	福岡市博多区	吉塚	吉塚	4	4LDK	100	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2020年第2四半期	未改装	NaN	7.518514
40029466	中古マンション等	NaN	40133	福岡県	福岡市中央区	谷	桜坂	5	4LDK	85	…	NaN	NaN	NaN	近隣商業地域	80.0	300.0	2017年第4四半期	改装済	NaN	7.623249
40014636	中古マンション等	NaN	40131	福岡県	福岡市東区	西戸崎	西戸崎	12	3LDK	75	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2021年第2四半期	改装済	NaN	7.113943

データコンペの概要

- 分析の手順
  1. 学習・評価用データのダウンロード
  2. データの確認
  3. 前処理・データ解析
  4. モデルの構築と評価

データコンペの概要

- 評価方法
  - 目的変数は, 取引価格（総額）について常用対数をとったもの.
  - 予測精度の評価は平均絶対誤差 (Mean Absolute Error, 以下 MAE とする):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|.$$

ここで, $\hat{y}_i$ は i 番目のデータにおける予測値, y_i は i 番目のデータにおける実測値である.

勾配ブースティング決定木の概要

- 目標は, 説明変数 $\boldsymbol{x}$ を入力したときに, 目的変数 y を出力する関数

$$y := F(\boldsymbol{x}; \{\rho_m, \boldsymbol{a}_m\}_1^M) = \sum_{m=1}^{M} \rho_m h_m(\boldsymbol{x}; \boldsymbol{a}_m)$$

  の予測である. ただし, M は決定木の本数, ρ_m は実数で重みを表し, 関数 $h_m(\boldsymbol{x}; \boldsymbol{a}_m)$ は $\boldsymbol{a}_m$ でパラメタづけられた回帰木, $\boldsymbol{a}_m$ は回帰木を特徴づけるパラメータである.

- この F のパラメータを決定する際に, 勾配ブースティングという手法を用いる.

勾配ブースティングの手法

- 勾配ブースティングは, 勾配降下法とブースティングを組み合わせた手法である.
- 勾配降下法: 関数の最小値を求める際に用いられるアルゴリズム. 勾配を最も減少させる方向に値を更新していく.
- ブースティング: アンサンブルの一種. 学習したモデルから次のモデルをつくり, モデル全体の予測値を出力する際に, 個々のモデルの予測値に重みをかけて足し合わせている.

記号の準備

- $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ を訓練データとする. ただし, 各 $i \in \{1, \cdots, n\}$ に対して, $\boldsymbol{x}_i$ を説明変数, y_i を目的変数とする.
- $L(a, b), a, b \in \mathbb{R}$ は損失関数で, 候補として,

$$|a - b|, \ (a - b)^2$$

がなどがある.

勾配ブースティングのアルゴリズム

アルゴリズム (勾配ブースティング [2])

1. $F_0(\boldsymbol{x}) = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \rho)$
2. m=1 から M に対して, 次を行う:
   1. $\tilde{y}_i = -\left[\dfrac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}, i = 1, \cdots, N$
   2. $\boldsymbol{a}_m = \underset{\boldsymbol{a},\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (\tilde{y}_i - \beta h(\boldsymbol{x}_i; \boldsymbol{a}))^2$
   3. $\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))$
   4. $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$

アルゴリズムの解説

アルゴリズム (勾配ブースティング [2])

1. $F_0(\boldsymbol{x}) = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \rho)$

- 初期値を設定する. $L(a,b) = |a-b|$ の場合は中央値,
  $L(a,b) = (a-b)^2$ の場合は平均値 (重心) が出力される.

アルゴリズムの解説

アルゴリズム (勾配ブースティング [2])

2. m=1 から M に対して, 次を行う:
   1. $\tilde{y}_i = -\left[\dfrac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}, i = 1, \cdots, N$

- 各 i に対して, 訓練データ i から定まる時点 $m-1$ での関数の勾配（最大減少量）を求める.

アルゴリズムの解説

アルゴリズム (勾配ブースティング [2])

2. m=1 から M に対して, 次を行う:
   2. $\boldsymbol{a}_m = \underset{\boldsymbol{a},\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (\tilde{y}_i - \beta h(\boldsymbol{x_i}; \boldsymbol{a}))^2$

- 求めた勾配に合うような決定木を 2 乗誤差基準で求める.
- このステップでは決定木の重みを保持しない.

アルゴリズムの解説

アルゴリズム (勾配ブースティング [2])

2. m=1 から M に対して, 次を行う:
   3. $\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))$

- 設定した損失関数の下, 最適な決定木の重みを求める.

アルゴリズムの解説

アルゴリズム (勾配ブースティング [2])

2. m=1 から M に対して, 次を行う:
    4. $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$

- 回帰木を更新する.

勾配ブースティングのアルゴリズム (再掲)

アルゴリズム (勾配ブースティング [2])

1. $F_0(\boldsymbol{x}) = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \rho)$
2. m=1 から M に対して, 次を行う:
   1. $\tilde{y}_i = -\left[\dfrac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}, i = 1, \cdots, N$
   2. $\boldsymbol{a}_m = \underset{\boldsymbol{a},\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (\tilde{y}_i - \beta h(\boldsymbol{x}_i; \boldsymbol{a}))^2$
   3. $\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))$
   4. $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$

他の決定木モデルとの違い

- GBDT と決定木
  - GBDT では, 各決定木は目的変数を出力するのではなく, 勾配 (残差) を出力している.
  - GBDT はブースティングによって精度が上がる代わりに, 決定木の特徴であった解釈性が失われる.
- GBDT とランダムフォレスト (以下 RF とする.)
  - GBDT ではアンサンブルにブースティングが行われているが, RF ではバギングが行われる.

→ ランダムフォレストは“並列に”, GBDT は“直列に”決定木を作っている.

XGboost の概略

- Extreme Gradient Boosting [7] の略であり, GBDT を扱うフレームワークである.
- 目的関数として, 損失関数に過学習防止のための正則化項を加えたものを用いている.
  - 正則化項には, 木の複雑さが大きくなると罰を与えるものが採用されている.
- 詳細は [6] や [7] を参照.

LightGBM の概略

- Light Gradient Boosting Machine の略で, XGboost と同じく, GBDT を扱うフレームワークである.
- XGboost と比較して決定木の構成精度をあまり落とすことなく, 高速で学習, 処理できるようになったのが最大の特徴である. ”Light(軽い=高速)”の由来にもなっている.
- リリース以降, 精度の高さと高速処理性からデータ分析コンペでの上位モデルに良く採用されている.

XGboost と LightGBM の違い

- XGboost に比べ, LightGBM の処理が速い理由として, 以下のことが知られている.
  - 決定木の構成を Level-Wise から Leaf-Wise に変更.
  - 決定木の分岐の決定の際に, GOSS[1] と EFB[2] というオリジナルの手法を用いている
- 詳細は Appendix を参照.

[1] Gradient-based One-Side Sampling の略.
[2] Exclusive Feature Bundling の略.

各フレームワークの性能比較の実験

- 実際に XGboost と LightGBM では, どれだけ性能の差があるのかについて実験を行った.
- 目的
  - 深さを変えた際の XGboost と LightGBM の学習時間とスコアの比較.
- パラメータの条件
  - 学習率 0.1, 評価指標 MAE, その他のパラメータは固定.
- 用いるデータ
  - nishika データコンペで配布された中古マンションのデータ.

各フレームワークの性能比較の実験

- 深さの設定
  - 5 から 10 の場合を調べる. なお, 深さの設定に合わせて, 決定木の本数は事前にパラメータを調整している.
- 各フレームワークの設定
  - XGboost のデフォルトの設定と LightGBM のデフォルトの設定.
- スコアの評価
  - 4 分割でのクロスバリデーションを行い, スコアの平均で評価.

実験結果: 学習時間

Table: 学習時間

深さ	木の本数	XGboost [s]	LightGBM [s]
5	13768	4301	101
6	8116	2984	53
7	5176	2232	29
8	3428	1688	20
9	2323	1294	14
10	1670	1045	11

- XGboost と LightGBM を比べると, XGboost の方が数十分から 1 時間単位で学習に時間がかかるのに対して, LightGBM は数十秒から 2 分弱で学習が終わったことがわかる.

実験結果: スコア

Table: スコア

深さ	木の本数	XGboost	LightGBM
5	13768	0.1467	0.1430
6	8116	0.1395	0.1404
7	5176	0.1387	0.1360
8	3428	0.1343	0.1352
9	2323	0.1340	0.1339
10	1670	0.1331	0.1344

- XGboost と LightGBM を比べると, スコアに関する有意な差はないように見える.

実験のまとめ

- この結果を見ると, LightGBM の方がタイムパフォーマンスが良いことがわかる.

→ 短時間に何度もモデルを改良しようと考え, 今回は LightGBM を採用した.

分析の手順 (再掲)

1. 学習・評価用データのダウンロード
2. データの確認
3. 前処理・データ解析
4. モデルの構築と評価

データの確認

- 訓練用データは 26 個の説明変数と目的変数の “取引価格 (総額)_log” から構成されている.
- テスト用データは 26 個の説明変数のみ構成されていて, 目的変数を予想する必要がある.

訓練用データの例 (再掲)

ID	種類	地域	市区町村コード	都道府県名	市区町村名	地区名	最寄駅：名称	最寄駅：距離（分）	間取り	面積（㎡）	…	前面道路：方位	前面道路：種類	前面道路：幅員（m）	都市計画	建ぺい率（%）	容積率（%）	取引時点	改装	取引の事情等	取引価格（総額）_log
40045006	中古マンション等	NaN	40136	福岡県	福岡市城南区	片江	福大前	14	2LDK	65	…	NaN	NaN	NaN	第２種住居地域	60.0	200.0	2022年第1四半期	未改装	調停・競売等	7.079181
40108482	中古マンション等	NaN	40203	福岡県	久留米市	諏訪野町	西鉄久留米	7	3LDK	75	…	NaN	NaN	NaN	商業地域	80.0	400.0	2016年第1四半期	未改装	NaN	7.255273
40162657	中古マンション等	NaN	40136	福岡県	福岡市城南区	別府	別府(福岡)	3	2LDK	60	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2008年第4四半期	未改装	NaN	7.146128
40108562	中古マンション等	NaN	40203	福岡県	久留米市	諏訪野町	花畑	13	3LDK	60	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2016年第4四半期	未改装	NaN	7.000000
40048209	中古マンション等	NaN	40137	福岡県	福岡市早良区	荒江	西新	16	2LDK	55	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2020年第3四半期	未改装	NaN	7.079181
40040759	中古マンション等	NaN	40135	福岡県	福岡市西区	拾六町	橋本(福岡)	28	3LDK	55	…	NaN	NaN	NaN	第２種住居地域	60.0	200.0	2014年第3四半期	改装済	NaN	7.079181
40054546	中古マンション等	NaN	40132	福岡県	福岡市博多区	住吉	博多	13	1K	25	…	NaN	NaN	NaN	商業地域	80.0	500.0	2015年第4四半期	未改装	NaN	7.079181
40025069	中古マンション等	NaN	40132	福岡県	福岡市博多区	吉塚	吉塚	4	4LDK	100	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2020年第2四半期	未改装	NaN	7.518514
40029466	中古マンション等	NaN	40133	福岡県	福岡市中央区	谷	桜坂	5	4LDK	85	…	NaN	NaN	NaN	近隣商業地域	80.0	300.0	2017年第4四半期	改装済	NaN	7.623249
40014636	中古マンション等	NaN	40131	福岡県	福岡市東区	西戸崎	西戸崎	12	3LDK	75	…	NaN	NaN	NaN	第１種住居地域	60.0	200.0	2021年第2四半期	改装済	NaN	7.113943

データの確認

- 説明変数のデータ型は object 型が 16 個, float 型が 10 個, int64 型が 1 個含まれている.
  - object 型の例: 都道府県名, 建築年
  - float 型の例: 容積率
  - int64 型の例: 市区町村コード
- データを見ると, 欠損値があり, また空白のカラムも存在している.
- LightGBM にデータを入力する際には, 文字データを数値にするなどの前処理が必要である.

データの前処理

- モデル構築に関係ないと思われる情報を消去する.
  - 訓練用データ, テスト用データ共に, 全て欠損値 (空白) のものを消去した.
  - 要素数が一種類しかないものも消去した.
  - 市区町村名と市区町村コードなどの同じ情報を表す説明変数は, どちらか一方を消去した.

データ前の処理

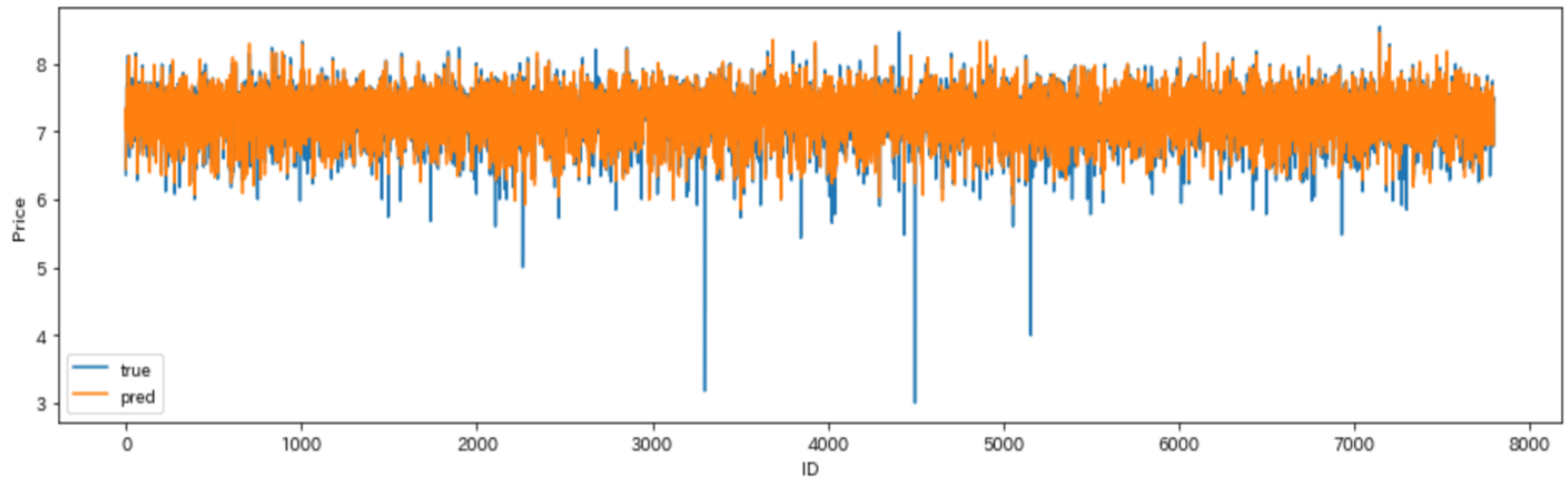
- 文字データである object 型を数値に変換する.
  - 建築年の具体的な数字に変換できるものは, 多少加工して数値に変換した.
  - 都道府県名などの単純に数値に変換できないものは, Category 型に変換した.

モデルの構築と評価

- 訓練データからモデルを作成するが, モデルの評価のために訓練データを更に検証データと学習用データに分割した.
- LightGBM でモデルを作成し, パラメータを適当に設定してモデルを回してみた結果, 精度は 0.081〜 0.083 だった.
- スコアを改善するため, Optuna[3] を用いてパラメータのチューニングを行った.

[3] ハイパーパラメータの最適化を自動化するためのソフトウェアフレームワーク.

予測結果とコンペの結果



- はずれ値はあるものの, 予測値と実際の値の差はあまりないように見え, 精度は 0.074 〜 0.076 程度だった.
- テストデータから予測値を求め提出したところ, 一番よかった結果の最終スコアは 0.074975(14 位/362 人) だった.

反省点

- 外れ値に関しての前処理は足りていない気がする: 相乗平均, Box-Cox 変換 (データの分布を正規分布に近づけるための変換) など一般のデータ変換を試せなかった.
- Public data の精度は valid data の精度より低いので, 過学習の可能性が高い.
- GBDT, RF 以外のモデルは試さなかった (多次元 AutoEncoder モデルなど).

今後の展望

- 他のモデルに対する理解を深める.
- 効果的なデータ処理に対する理解を深める.
- 他のテーマのコンペや他サイトのコンペへの参加 (Kaggle, SIGNATE).

参考文献 I

[1] Breiman, L., Friedman, J. H. , Olshen, R. A. , Stone, C. J. (1984). Classification and Regression Trees. Wadsworth.

[2] Friedman, J.H. (2001) Greedy Function Approximation: A Gradient Boosting Machine, The Annals of Statistics , Vol. 29, No. 5, pp. 1189-1232.

[3] DataScienceHub. 回帰の決定木アルゴリズムを完全図解する【機械学習入門 28】. https://datawokagaku.com/decision_tree/. 更新日 2022 年 5 月 9 日. 閲覧日 2023 年 11 月 9 日.

[4] ビショップ, C.M. (2012). パターン認識と機械学習, 元田浩, 栗田多喜夫, 樋口知之, 松本裕治, 村田昇監訳, 丸善出版.

[5] scikit-learn(n.d.). 1.10. Desicion Trees. https://scikit-learn.org/stable/modules/tree.html. 閲覧日 2023 年 11 月 11 日.

参考文献 II

[6] Chen, T., Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining pp. 785-794.

[7] XGboost Documentation(n.d.). Introduction to Boosted Trees. https://xgboost.readthedocs.io/en/stable/tutorials/model.html. 閲覧日 2023 年 11 月 15 日.

[8] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, 30.

[9] Microsoft Corporation(n.d.). LightGBM.https://lightgbm.readthedocs.io/en/stable/index.html. 閲覧日 11 月 20 日

[10] DataScienceHub. LightGBM を超わかりやすく解説 (理論＋実装)【機械学習入門 33】.https://datawokagaku.com/lightgbm/. 更新日 2022 年 5 月 19 日. 閲覧日 11 月 20 日

参考文献 III

[11] creative commons. 決定木.https://axa.biopapyrus.jp/machine-learning/decision-tree/. 更新日 2019 年 6 月 30 日. 閲覧日 11 月 20 日.

[12] HatenaBlog. 【論文紹介】LightGBM: A Highly Efficient Gradient Boosting Decision Tee. https://nnkkmto.hatenablog.com/entry/2020/12/15/000000. 更新日 2020 年 12 月 15 日. 閲覧日 11 月 20 日

[13] Nishika(n.d.). 中古マンション価格予測 夏の部. https://competition.nishika.com/competitions/mansion_2023summer/summary#description. 閲覧日 2023 年 11 月 23 日.

GBDT の補足

- 以下では, GBDT の理論に欠かせない決定木の補足を行う.

回帰木

- 決定木には分類木と回帰木がある.
- 回帰木は基本的に, 目的変数を入力すると, その目的変数に対する説明変数の予測値を出力する.
- 今回は価格の予想なので, 回帰木を用いる.
- 次のスライド以降では, CART(Classification and Desicion Trees[1]) という決定木の構成手法を紹介する.

準備

$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ を訓練データとする. ただし, 各 $i \in \{1, \cdots, n\}$ に対して, $\boldsymbol{x}_i$ を説明変数, y_i を目的変数とする. また, 写像 $\boldsymbol{x}_i \mapsto y_i$ を $y(\boldsymbol{x}_i)$ と表す. また, $R(j)$ を

$$R(j) = \frac{1}{|V_j|} \sum_{x \in V_j} L(y(x), \bar{y}_j)$$

と定義する. ただし, V_j は j 番目のノードを表し,

$$\bar{y}_j = \frac{1}{|V_j|} \sum_{x \in V_j} y(x)$$

である. また, $L(y(x), \bar{y}_j)$ はロス関数で, 候補として,

$$|y(x) - \bar{y}_j|, \ (y(x) - \bar{y}_j)^2$$

がなどがある.

準備

ノードの分割全体の集合を $\mathcal{S}$ とする. また, $\Delta R(t)$ を

$$\Delta R(t, s) = R(t) - R(t_L) - R(t_R)$$

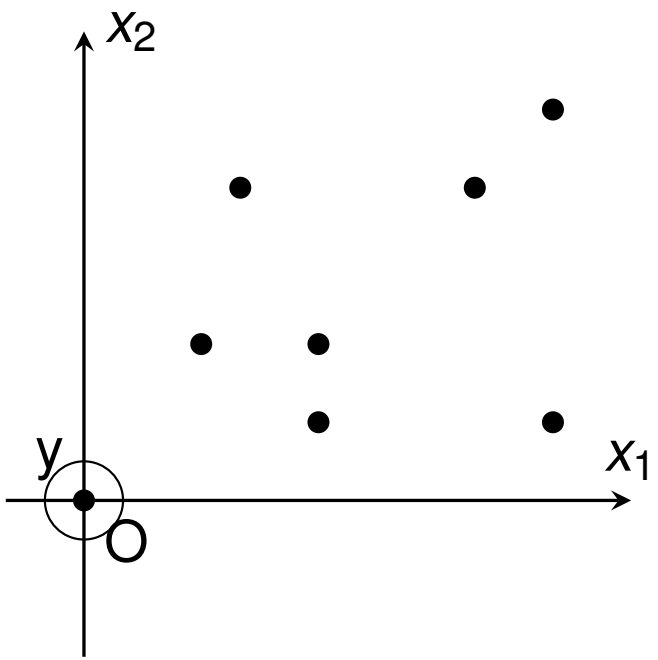
で定義する. ただし, t_L, t_R は分割 $s \in \mathcal{S}$ を決めた時に定まる t の子ノードの番号である. また, t_L, t_R には $\{1, \cdots, 2^m - 1\}$ の中から使われてない自然数のうち最小のものと 2 番目に最小のものを t_L. t_R に割り当てる.

CART

アルゴリズム (CART)

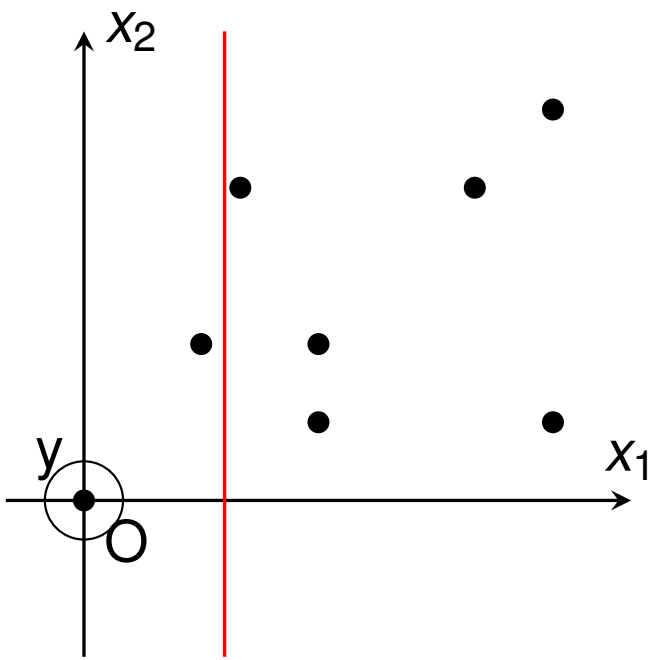
1. for $i = 1$ to $2^m - 1$
   1. $s = \underset{s' \in S}{\mathrm{argmax}} \Delta R(i, s')$
2. for $i = 2^m$ *to* $2^{m+1} - 1$
   1. $j = i - 2^m + 1$
   2. $V'_j = V_i$
   3. $\beta_j = \bar{y}_i$

イメージ



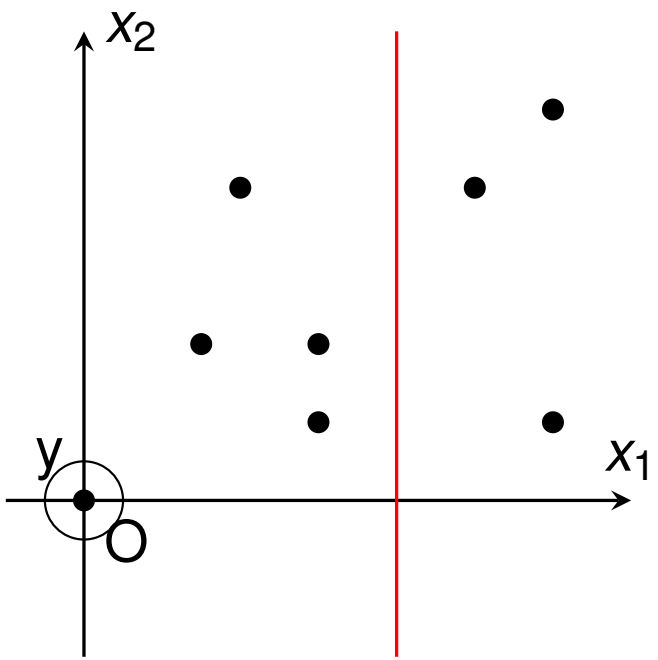
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



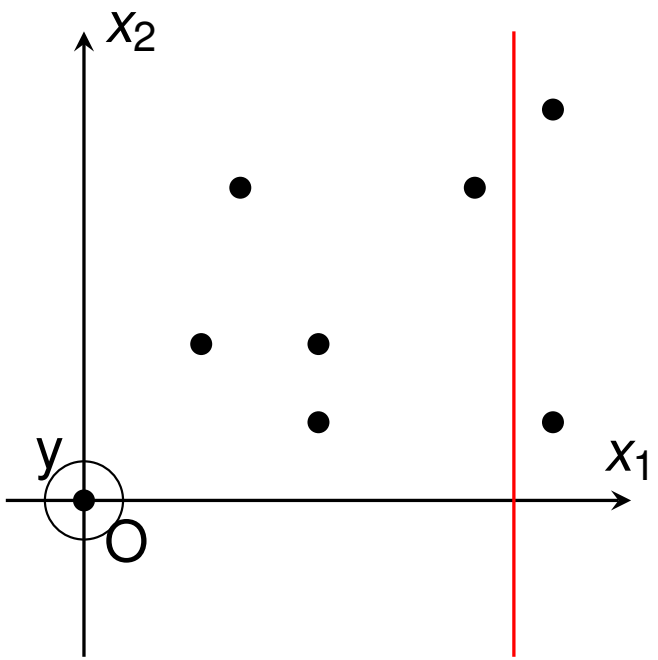
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



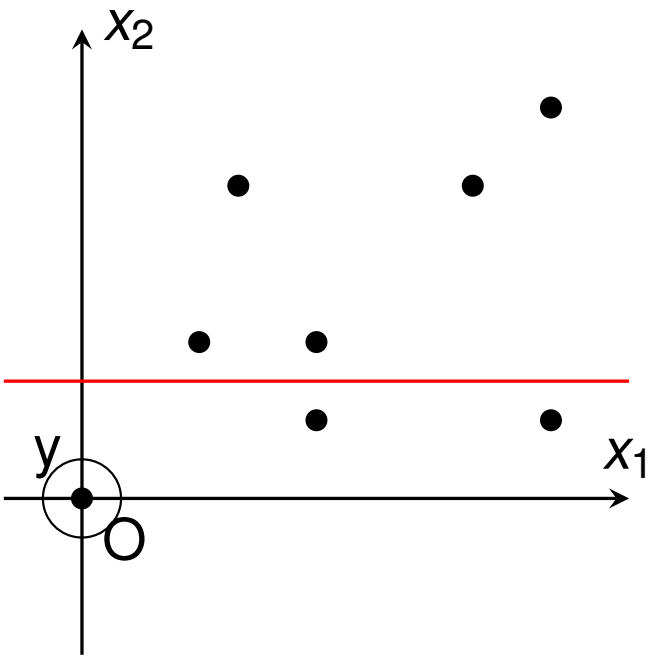
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



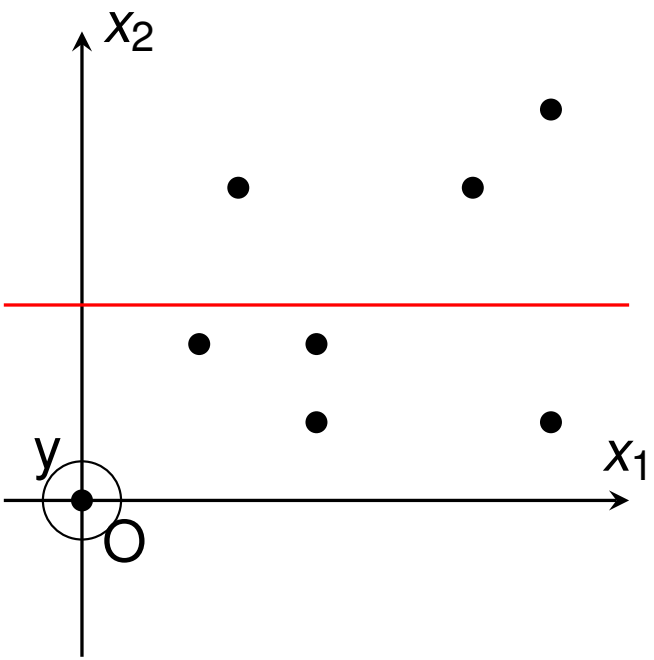
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



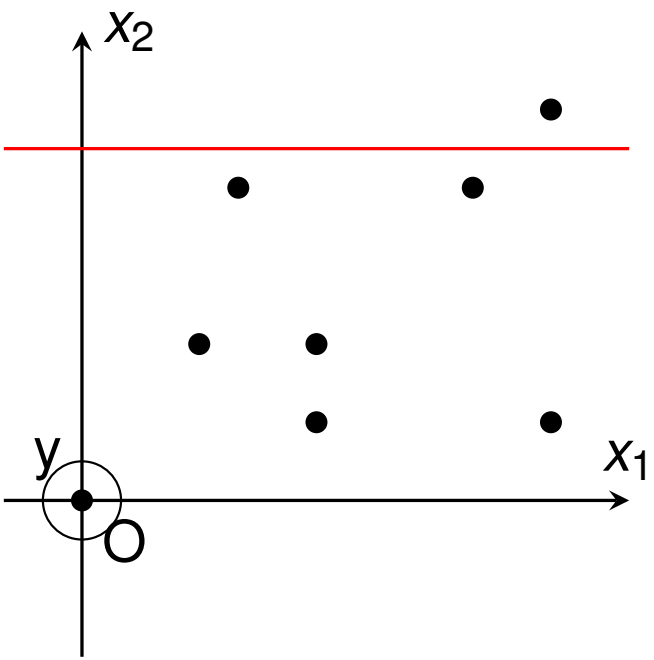
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



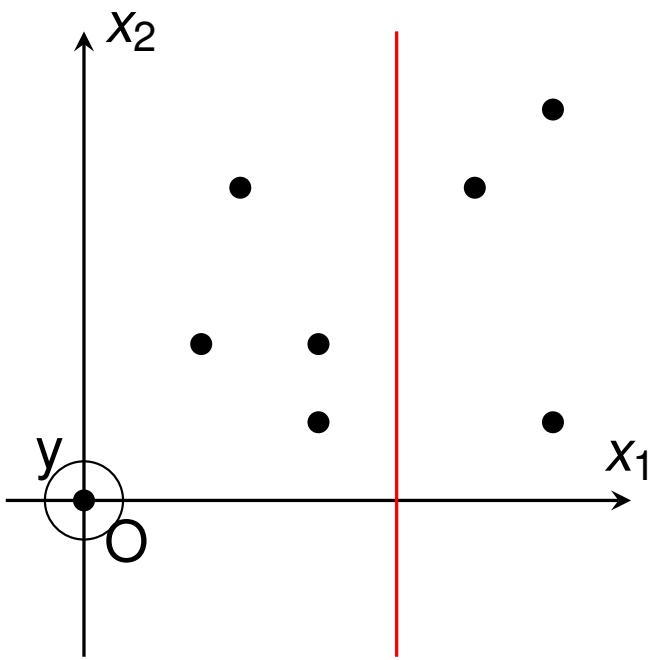
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



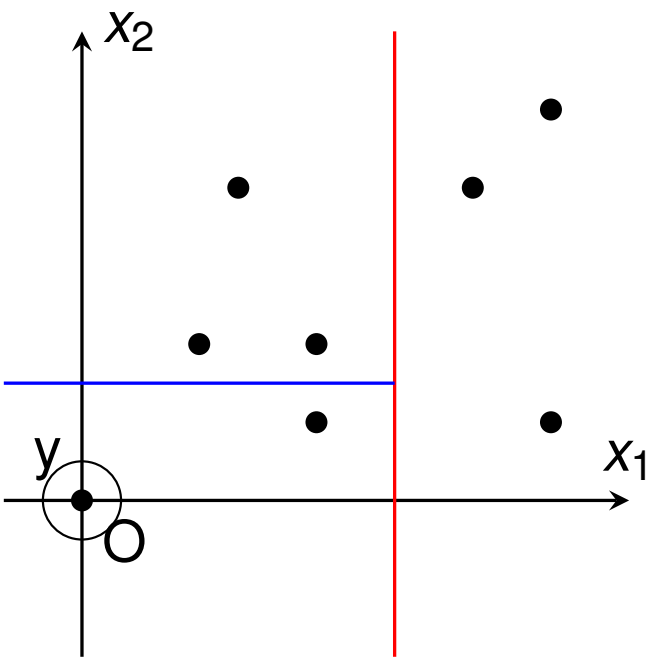
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



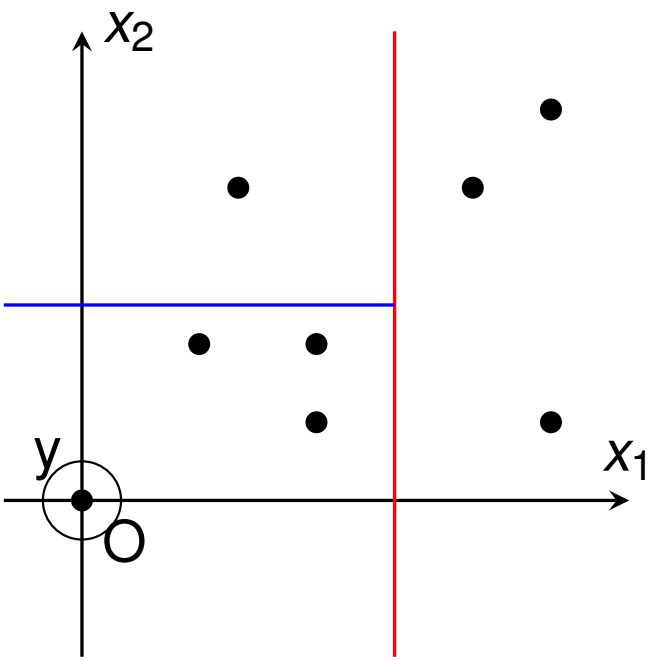
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



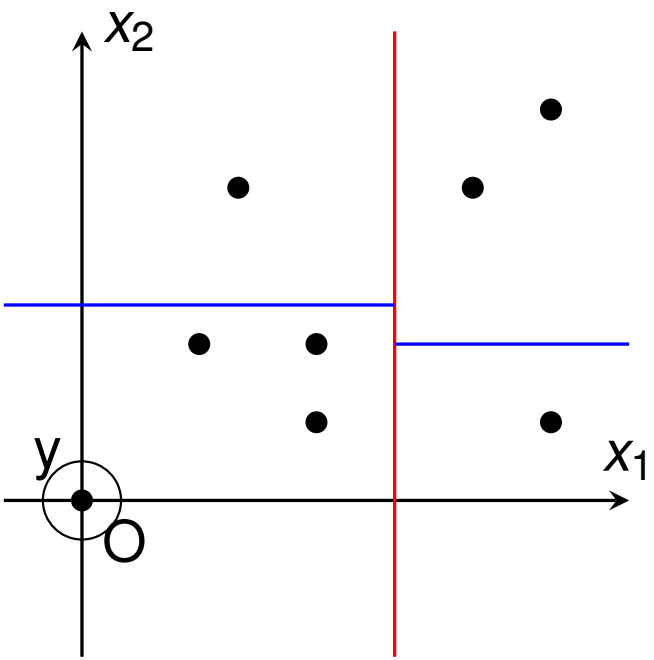
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



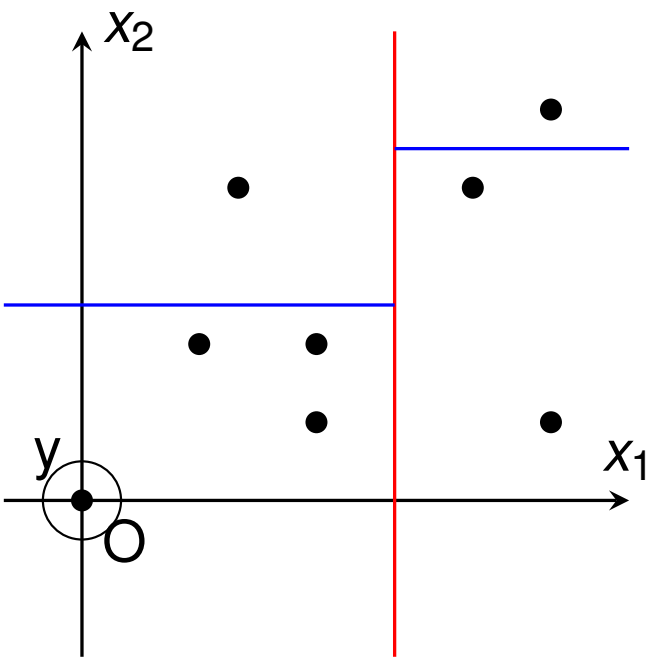
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



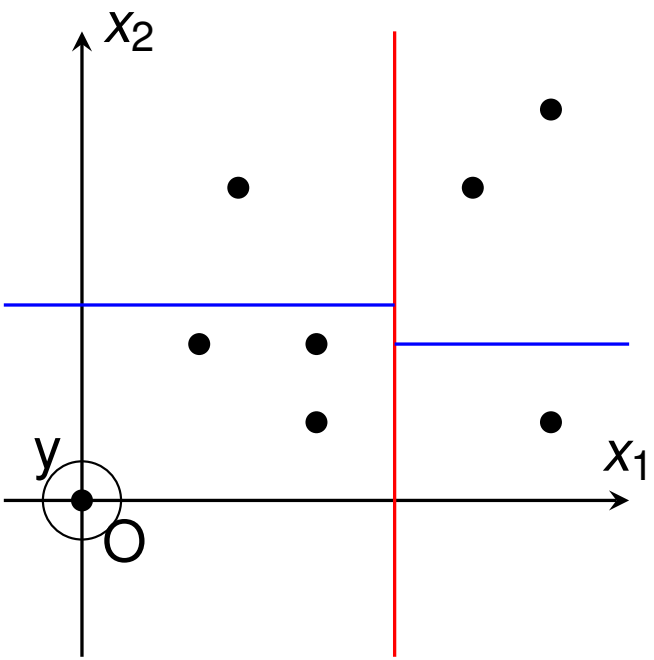
1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

イメージ



1. 訓練データを 2 つの領域に分け, 分割前と分割後の誤差の変化を計算する.
2. 変化の最も大きくなるような分割を採用する.
3. 深さ m まで 1 と 2 を繰り返す.
4. 分けられた領域ごとに, 領域内に含まれる正解ラベルの平均を予測値として出力する.

注意

- 予測された回帰木を $\widehat{F}(\boldsymbol{x})$ とすると,

$$\widehat{F}(\boldsymbol{x}) = \sum_{j=1}^{2^m} \beta_j 1_{V_j'}(\boldsymbol{x})$$

  と表せる.

- 全探索で分割の候補を探すので, 時間がかかる.

- 計算量の削減や過学習の防止のために, 実際には刈り込み (pruning) が行われているが, 今回は略した.

LightGBM の補足

- 以下では, LIghtGBM の高速処理に対する補足をする.

決定木の構成方法

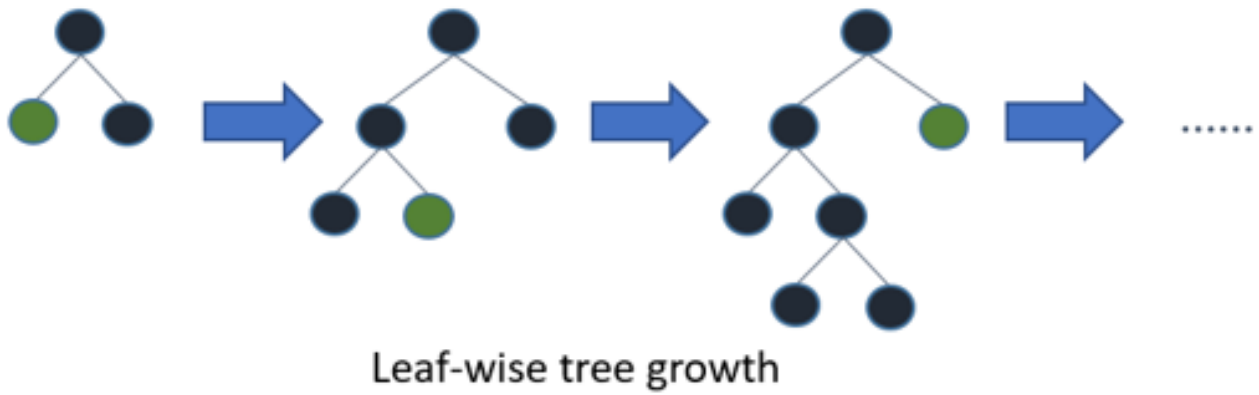
- Level-Wise: 決定木を深さごとに成長させていく手法. XGboost で採用されている.

Figure: Level-Wise のイメージ [9]

決定木の構成方法

- Leaf-Wise: 選択的に決定木を成長させる手法. LightGBM で採用されている.

Figure: Leaf-Wise のイメージ [9]



Leaf-Wise

- Leaf-Wise ではすべてのノードと特徴量の組み合わせで事前に情報利得を算出し, 情報利得が最大となるノードと特徴量で次の分割を実行する.
  - 情報利得とは, 親のノードの不純度とその子ノードの不純度の差のこと.
  - 不純度とは, 1 つのノードに異なるクラスのサンプルが含まれる割合を数値化した値のこと.

  → 情報利得は分割の前後を比較した際にどれだけデータを適切に分割できたかを数値として示す.

GBDT の処理時間

- GBDT において時間を要する工程の 1 つとしては, 全数探索を行う決定木の分岐の決定がある.
  - よって, 処理時間は特徴数 (列) とインスタンス数 (行) に比例する.

→ 特徴数とインスタンス数を精度を落とさないようにそれぞれうまくデータを削除・結合できれば, 計算効率が改善できる.

計算の効率化

- インスタンス数を減らす工夫として, Gradient-based One-Side Sampling(GOSS, 勾配ベース片側サンプリング) を行っている.
- 特徴数を減らす工夫として, Exclusive Feature Bundling(EFB, 排他的特徴バンドル) を行っている.

GOSS の発想

- GBDT では各インスタンスに勾配 (残差) を与えているが, 勾配の絶対値が小さなインスタンスは既に十分学習しているとみなせる.
- つまり, このようなデータは学習上, 重要性が低いデータとみなすことができる.
- 逆に勾配の絶対値が大きなインスタンスは重要性が高いデータとみなせる.

GOSS の仕組み

- そこで、GOSS では次のように学習対象となるインスタンス数を次のように絞り込むこととしている. $(0 < a, b < 1)$
  1. 勾配の絶対値に従って降順でインスタンスを並び替え、上位 a%すべてを選択する.
  2. 残されたインスタンスの b%をランダムサンプリングし、(1-a)/b を乗じて重みづけをおこなう.

GOSS の補足

- 2 のサンプリング後の処理はサンプリングによる勾配総和の減少効果を取り除く狙いがある.
- 処理の後のデータから元の情報利得が近似的に求まることが知られており、 このことから GOSS の有用性が担保されている.

EFB の発想

- 特徴数が多いデータ (高次元データ) を取り扱う時、ほとんどが 0(つまり「疎」) となることがよくある.
- このような場合, 同じインスタンス内で同時に非ゼロ値を取りずらい特徴量同士は束 (bundle) として一つの特徴量と同等の扱いが可能になる.

EFB の仕組み

1. 非ゼロ値の重複許容量を定める.
2. 許容量を基準に最適な特徴量らを選択する.
3. 一つの特徴量として結合する (束ねる).

EFB の問題

- 問題 1: どの特徴量を選択するか.
- 問題 2: どのように選択された特徴量らを束ねるか.

特徴量の選択への対応

- グラフ理論の議論に基づいて次のように処理される:
  1. 特徴をノード, 重複度を重み付きエッジとしたグラフを構成する.
  2. エッジから算出されるノードの重みの総和で降順にソートする.
  3. 降順に既存の束にまとめるか, 新規に束を構成するか判定する.

特徴量の束ね方への対応

- 束から元の特徴の値が識別できるようにする.
- 次のスライドで具体例を挙げる.

例: 非衝突 (非ゼロ値を同時に取らない) の場合

- 特徴 A が $[0, 10)$, 特徴 B が $[0, 20)$ の値をとるとする.
- この時, 特徴 B の各値を+10(offset) とした $[10, 30)$ となる特徴量 B' を考えられる.
- ここで A と B' は取りえる値の範囲が被らないので, 統合しても識別可能である.
- また, B と B' の対応がわかっているので, 統合後から A, B の復元が可能である.