勾配ブースティング
Gradient Boosting

明治大学 理工学部 応用化学科
データ化学工学研究室　金子 弘昌

勾配ブースティングとは？

✓アンサンブル学習の一つ

✓ブースティングの一つ

✓クラス分類でも回帰でも可能

✓クラス分類手法・回帰分析手法は何でもよいが、
基本的に決定木を用いる

- Gradient Boosting Decision Tree (GBDT)

✓クラス分類モデルで誤って分類されてしまったサンプルや、
回帰モデルで誤差の大きかったサンプルを、改善するように
(損失関数の値が小さくなるように) 新たなモデルが構築される

準備: 損失関数 (Loss Function) [回帰]

サンプルごとに、モデルが予測をミスした量を計算する関数 $L(y^{(i)}, f(\mathbf{x}^{(i)}))$

回帰係数の場合

$$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \frac{1}{2}\left(y^{(i)} - f\left(\mathbf{x}^{(i)}\right)\right)^2$$

$$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \left|y^{(i)} - f\left(\mathbf{x}^{(i)}\right)\right|$$

・・・など、いろいろとあります

$y^{(i)}$: i 番目のサンプルにおける目的変数の値
$\mathbf{x}^{(i)}$: i 番目のサンプルにおける説明変数ベクトル
f : 回帰モデル
$f(\mathbf{x}^{(i)})$: i 番目のサンプルにおける目的変数の推定値

準備: 損失関数 (Loss Function) [分類]

サンプルごとに、モデルが予測をミスした量を計算する関数 $L(y^{(i)}, f(\mathbf{x}^{(i)}))$

クラス分類の場合も
いろいろあります

Adaboost におけるサンプルの重み $w^{(i)}$
Adaboost や $w^{(i)}$ の詳細はこちら
https://datachemeng.com/adaboost/

特に決定木のときは交差エントロピー誤差関数

$$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = -\sum_{k=1}^{K} p_{j,k} \ln p_{j,k}$$

K : クラスの数
$p_{j,k}$: サンプル i が含まれる
葉ノード j における、
クラス k のサンプルの割合

$y^{(i)}$: i 番目のサンプルにおけるクラス
$\mathbf{x}^{(i)}$: i 番目のサンプルにおける説明変数ベクトル
f : クラス分類モデル
$f(\mathbf{x}^{(i)})$: i 番目のサンプルにおける推定されたクラス

準備: 損失関数 (Loss Function) [分類]

サンプルごとに、モデルが予測をミスした量を計算する関数 $L(y^{(i)}, f(\mathbf{x}^{(i)}))$

K クラスを分類する多クラス分類のとき、クラスごとにK 種類の
y (1 or −1) の変数を作成して、K 個の二クラス分類モデルを作成
(あるクラスにおいて、対象のサンプルが 1、それ以外のサンプルが −1)

k 番目の二クラス分類モデルを f_k とすると、

$$p_k\left(\mathbf{x}^{(i)}\right) = \frac{\exp\left(f_k\left(\mathbf{x}^{(i)}\right)\right)}{\sum_{j=1}^{K}\exp\left(f_j\left(\mathbf{x}^{(i)}\right)\right)}$$

$$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = -\log p_c\left(\mathbf{x}^{(i)}\right)$$

c : $y^{(i)}$ の実際のクラスに対応
する二クラス分類モデルの番号

準備: 損失関数の勾配 [回帰]

損失関数の勾配: $L(y^{(i)}, f(\mathbf{x}^{(i)}))$ を $f(\mathbf{x}^{(i)})$ で偏微分して、
マイナスの符号をつけたもの

“勾配” ブースティングの名前の由来

例) $$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \frac{1}{2}\left(y^{(i)} - f\left(\mathbf{x}^{(i)}\right)\right)^2$$

$$\Rightarrow \quad -\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)} = y^{(i)} - f\left(\mathbf{x}^{(i)}\right)$$

準備: 損失関数の勾配 [分類]

損失関数の勾配: $L(y^{(i)}, f(\mathbf{x}^{(i)}))$ を $f(\mathbf{x}^{(i)})$ で偏微分して、
マイナスの符号をつけたもの

“勾配” ブースティングの名前の由来

例) $$L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = -\log p_m\left(\mathbf{x}^{(i)}\right)$$

k 番目の二クラス分類モデルでは、

$$\Rightarrow \quad -\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f_k\left(\mathbf{x}^{(i)}\right)} = q - p_k\left(\mathbf{x}^{(i)}\right)$$

q : k が $y^{(i)}$ の実際のクラスに対応すれば $q = 1$,
そうでなければ $q = 0$

決定木の勾配ブースティングのアルゴリズム

決定木の詳細はこちら https://datachemeng.com/decisiontree/

✓最初の決定木モデル $f^{(0)}$ をいつも通り構築する

✓$m = 1, 2, \ldots, M$ として以下を繰り返す

- サンプル i ごとに損失関数の勾配を計算
- 損失関数の勾配を y として、決定木モデルを構築
- 決定木モデルの葉ノードを $R_{m,j}$ (j は葉ノードの番号) とする
- 葉ノードごとに、以下を最小化する $\gamma_{m,j}$ を計算

$$\sum_{\mathbf{x}^{(i)}\text{が含まれる}R_{m,j}} L\left(y^{(i)}, f^{(m-1)}\left(\mathbf{x}^{(i)}\right) + \gamma_{m,j}\right)$$

- 以下のようにモデル $f^{(m)}$ を計算 (η は学習率)

$$f^{(m)}(\mathbf{x}^{(i)}) = f^{(m\text{-}1)}(\mathbf{x}^{(i)}) + \eta \ (\mathbf{x}^{(i)} \text{ が含まれる } R_{m,j} \text{ における } \gamma_{m,j})$$

Python で勾配ブースティングを実行するには

✓scikit-learn の gradient boosting

- 回帰: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
- 分類: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

✓XGBoost (eXtreme Gradient Boosting)

- https://xgboost.readthedocs.io/en/latest/python/python_intro.html

✓LightGBM (Light Gradient Boosting Model)

- https://lightgbm.readthedocs.io/en/latest/

・・・など

- すべて決定木に基づく勾配ブースティング
- XGBoost, LightGBM には、オーバーフィッティングを防いだり、計算速度を上げるための工夫あり

XGBoost の主な工夫

決定木モデルを構築するときの評価関数 E に以下の式を加える

$$\gamma T + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

T : 最終ノードの数
$\mathbf{w}$: すべての葉ノードの値 (葉ノードのサンプルの y の平均値) が格納されたベクトル
γ, λ : ハイパーパラメータ

⇨ ✓ ノード数を小さくする
✓ モデルの更新を小さくする ⇨ オーバーフィッティングの低減

さらに、m での決定木の構築に、m-1 での決定木の勾配情報を利用して計算速度向上も

決定木の詳細はこちら https://datachemeng.com/decisiontree/

LightGBM の主な特徴

✓Gradient-based One-Side Sampling (GOSS)

- m 回目の学習のとき、勾配の絶対値の大きな $a \times 100\ \%$ のサンプルと、それ以外のサンプルから $b \times 100\ \%$ をランダムに選択したサンプルのみを用いる
- ランダムに選択されたサンプルの勾配は $(1\text{-}a)/b$ 倍して利用

✓Exclusive Feature Bundling (EFB)

- 0 でない値をもつサンプルのかぶりが少ない変数は、一緒にして “bundle” として扱う
  - 例えば 0 から 10 の値をもつ変数 A と 0 から 20 の値をもつ変数 B があるとき、B に一律に 10 足して、A と B とを合わせたものが bundle
- 変数からではなく、bundle から決定木を作成
- ヒストグラムで値をざっくり分けてから決定木の分岐を探索

⇨ オーバーフィッティングの低減 & 計算速度向上

デモの Python プログラムあります！

こちら https://github.com/hkaneko1985/gradient_boosting へどうぞ！

✓XGBoost や LightGBM を利用したい方は、まずはそれぞれを
インストールする必要があります (Anaconda 利用を想定)

- XGBoost: https://anaconda.org/anaconda/py-xgboost
- LightGBM: https://anaconda.org/conda-forge/lightgbm

✓demo_of_gradient_boosting_classification_default_hyperparam.py,
demo_of_gradient_boosting_regression_default_hyperparam.py
ではデフォルトのハイパーパラメータで実行されます。method_flag で
お好きな手法の番号にして実行してください

✓demo_of_gradient_boosting_classification_with_optuna.py,
demo_of_gradient_boosting_regression_with_optuna.py では
optuna でハイパーパラメータの最適化をしてから各手法が実行されます。
optunaをインストールする必要があります

- optuna: https://optuna.org/

データセットを変えて、プログラムをご利用ください[12]

✓optuna を用いたハイパーパラメータの最適化はこちらの examples
https://github.com/pfnet/optuna/tree/master/examples
を参考にしました

- ただ、デフォルトのパラメータでも良好なモデルを構築できたり、ベイズ最適化するよりテストデータの推定結果がよくなったりすることもあります。デフォルトのパラメータとベイズ最適化したパラメータとで両方比較するとよいです。

✓今回のデモのプログラムでは以下のデータセットを利用しています

- 回帰: ボストンのデータセット
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
- 分類: あやめのデータセット
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

✓x (説明変数), y (目的変数) のデータを変えるだけで
(19, 20行目あたり)、デモのプログラムをそのまま利用できます。
ご活用ください！

参考文献

- ✓A. Natekin, A. Knoll, Gradient boosting machines, a tutorial, Front. Neurobot., 7, 1-21, 2013 https://doi.org/10.3389/fnbot.2013.00021
- ✓T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data mining, Inference, and Prediction, Second Edition, Springer, 2009 https://web.stanford.edu/~hastie/ElemStatLearn//
- ✓T. Chen, C. Guestrin, XGBoost: A Scalable Tree Boosting System, arXiv:1603.02754, 2016 https://arxiv.org/abs/1603.02754
- ✓G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T. Y. Liu, LightGBM: A Highly Efficient Gradient Boosting Decision Tree, NIPS Proceedings, 2017 https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree