和文タイトル

English Title

著者 1 氏名 [1*]　著者 2 氏名 [1,2]
Author1[1]　Author2[1,2]

[1] 所属機関 1 (和文)
[1] Affiliation 1 (English)
[2] 所属機関 2
[2] Affiliation 2

Abstract:

1 はじめに

2 既存手法 (gBoost)

本章では、既存手法である gBoost[1] について説明する。gBoost は線形計画問題で定式化されたブースティング手法である LPBoost[2] と、部分グラフの探索アルゴリズムを組み合わせたグラフの分類、回帰モデルである。本研究ではグラフの分類問題を扱うため、以下では、グラフの線形分類モデル、LPBoost、特徴探索アルゴリズム、と順を追って説明する。

2.1 グラフ線形分類モデル

まず、グラフに対する線形分類モデルについて説明する。一般的に機械学習の線形モデルは、n 個の特徴 $x_1, x_2, \ldots, x_n$ に対して、$f(\boldsymbol{x}) = a_1x_1 + a_2x_2 + \cdots + a_nx_n + b$ とし、2 値分類（+1 or -1）の場合、$f(x) \geq 0$ の時に 1、$f(x) \leq 0$ の時に-1 と予測する。グラフ分類問題に用いる特徴は様々なものが考えられるが、gBoost ではグラフデータに含まれる部分グラフの有無を特徴として用いる。入力として l 個のグラフとクラスラベル $\{G_n, y_n\}_{n=1}^{l} (y_n \in \{+1, -1\})$ が与えられた時、入力データに少なくとも 1 回は出現する全ての部分グラフを $\mathcal{T}$ とする。この時グラフ G_n は $|\mathcal{T}|$ 次元のベクトル $\boldsymbol{x}_n (\boldsymbol{x} = \mathbb{I}(t \subseteq G_n), \forall t \in \mathcal{T})$ として表現することができる。$I(\cdot)$ は括弧内が真なら 1、偽なら 0 を返す関数である。例を図 1 に示す。
この時、各特徴に対して仮説 (hypotheses) と呼ばれる、弱学習器を以下で定義する。

$$h(\boldsymbol{x}; t, \omega) = \omega(2x_t - 1). \tag{1}$$

$\omega \in \Omega = \{-1, 1\}$ はパラメータである。この弱学習器を用いて、gBoost では以下の線形分類モデルを構築する。

$$f(\boldsymbol{x}) = \sum_{(t,\omega) \in \mathcal{T} \times \Omega} \alpha_{t,\omega} h(\boldsymbol{x}; t, \omega). \tag{2}$$

$\alpha_{t,\omega}$ は各弱学習器の重みであり、$\sum_{(t,\omega) \in \mathcal{T} \times \Omega} \alpha_{t,\omega} = 1, \alpha_{t,\omega} \geq 0$ を満たす。

2.2 LPBoost

gBoost では、LPBoost で定式化された以下の線形計画問題を主問題として解くことで、先ほど定義した線形モデルの重み $\alpha_{\{t,\omega\}}$ の値を導く。

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\xi}, \rho} -\rho + D \sum_{n=1}^{l} \xi_i \tag{3}$$

$$s.t. \sum_{(t,\omega) \in \mathcal{T} \times \Omega} y_n \alpha_{t,\omega} h(\boldsymbol{x}_n; t, \omega) + \xi_n \geq \rho, \tag{4}$$

$$\xi_n \geq 0,$$
$$n = 1, \ldots, l,$$
$$\sum_{(t,\omega) \in \mathcal{T} \times \Omega} \alpha_{t,\omega} = 1,$$
$$\alpha_{t,\omega} \geq 0.$$

ここで、ρ はソフトマージン、$D = \frac{1}{\nu l}, \nu \in (0, 1)$ であり、ν は誤分類に対するコスト制御のパラメータである。一般に、変数 $\boldsymbol{\alpha}$ の数は部分グラフの総数であるため膨大であり、主問題を解くことは困難である。したがって、主問題と等価な以下の双対問題を扱う。

*連絡先： (所属機関名)
(所属機関住所)
E-mail:

$$\min_{\boldsymbol{\lambda},\gamma} \gamma \tag{5}$$

$$s.t. \sum_{n=1}^{l} \lambda_n y_n h(\boldsymbol{x}_n; t, \boldsymbol{\omega}) \leq \gamma, \tag{6}$$

$$\forall (t, \boldsymbol{\omega}) \in \mathcal{T} \times \Omega,$$

$$\sum_{n=1}^{l} \lambda_n = 1,$$

$$0 \leq \lambda_n \leq D,$$

$$n = 1, \ldots, l.$$

双対問題は、主問題に比べ制約式の数は多くなるが列生成法を用いることで解を得ることが可能となる。列生成法は、制約なしの状態から始め、現制約を最も違反する制約を追加し、最適化問題を解くという過程を繰り返すことで解を得る手法である。最も違反する制約を見つける問題は以下の最大化問題と等価である。

$$(t^*, \boldsymbol{\omega}^*) = \arg\max_{t \in \mathcal{T}, \boldsymbol{\omega} \in \Omega} \sum_{n=1}^{l} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \boldsymbol{\omega}). \tag{7}$$

これに対して最大化する目的関数を gain と呼び以下の式で定義する。

$$g(t, \boldsymbol{\omega}) = \sum_{n=1}^{l} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \boldsymbol{\omega}). \tag{8}$$

もし以下を満たすような弱学習器が存在しない場合、現在の解が双対問題の最適解となる。

$$\sum_{n=1}^{l} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \boldsymbol{\omega}) > \gamma^{(k)}. \tag{9}$$

経験的に終了付近の制約追加による重みの更新は僅かであるため、以下のように終了条件を緩和することで、計算を効率化する。

$$\sum_{n=1}^{l} \lambda_n^{(k)} y_n h(\boldsymbol{x}_n; t, \boldsymbol{\omega}) > \gamma^{(k)} + \varepsilon. \tag{10}$$

双対問題の終了条件緩和により得られる主問題の解は、V^* を緩和なしでの最適解とすると、$V \leq V^* + \varepsilon$ を満たす。

2.3 特徴探索アルゴリズム

特徴探索は重複無しに全部分グラフを列挙した探索空間である DFS コード木 [3] を用いる。DFS コード木は、全部分グラフをノードに含む他に、子が親のスーパーグラフとなるという特徴を持つ。この探索木を深さ優先的にたどり、各ノードにおいて gain を計算することで、最適な部分グラフを探索する。一般に、部分グラフの総数は膨大となるため全てのノードを探索するためには相当なコストを要する。しかし、子が親のスーパーグラフとなる特徴を利用すると、bound(子孫ノードでの gain の上限値) を計算することができるため、安全な枝刈りを行うことが可能である。bound の計算、枝刈り可能条件を以下に示す。

現探索部分グラフを t、そのスーパーグラフを t' とすると、bound の値 $b(t)$ は以下の式で計算できる。

$$b(t) = \max\{2 \sum_{\{n|y_n=+1, t \subseteq G_n\}} \lambda_n^{(k)} - \sum_{n=1}^{l} y_n \lambda_n^{(k)},$$

$$2 \sum_{\{n|y_n=-1, t \subseteq G_n\}} \lambda_n^{(k)} - \sum_{n=1}^{l} y_n \lambda_n^{(k)}\}. \tag{11}$$

枝刈り可能条件は、現探索までの gain の最大値を g^* とすると以下となる。

$$g^* > b(t). \tag{12}$$

この枝刈りが有効に働くため、gBoost アルゴリズムは探索空間に対して効率よくモデルを構築することができる。

3 提案手法

既存手法の gBoost では探索木上を深さ優先的に厳密探索する。しかし、入力グラフ集合に対して、個々のグラフの大きさや全体数の増加に伴い、存在しうる部分グラフの総数は急激に増加するため、厳密探索におけるコストは莫大なものである。問題によってモデル構築が困難なものも存在する。本稿では、厳密探索を行わず、乱数を用いて探索をランダム化する手法を提案する。

まず初めに、本提案手法で用いる *flow* の概念を説明する。

1. ある定数 k を定める。
2. 探索木の根ノードから探索を開始する。
3. 現探索ノードにおける子ノードを列挙する。
4. 列挙した m 個の各子ノード i に対して、確率 $p(i)(1 \leq i \leq m)$ を定める。
5. 定めた確率に基づいて、入力定数 k を各子ノードに割り振る。
6. 各子ノードにおいて、割り振られた数を入力定数とし 3〜5 を再帰的に行う。
7. 探索が末端ノードに到達した場合、入力定数が 1 以上であっても再帰を終了する。(ここで末端ノードとは子ノードを持たないノード、枝刈り可能なノード、指定したグラフの大きさを上回る部分グラフを表現するノードのことを指す。)

以上の探索を *k flow* 探索と定義する。Algorithm1 に擬似コードを示す。

Algorithm 1 探索のランダム化

```
1: procedure 最適な部分グラフ、あるいはその共起
2:     グローバル変数 : g*, ω*, p*, pc*
3:     最適な部分グラフ g*, ω*, p* と Top-k 部分グラフを探索  ▷ gBoost 法の探索
4:     for all p ∈ Top-k 部分グラフ do
5:         project(p)
6:     end for
7: end procedure
8: function TMP(p,t)
9:     if b(t) < g* then
10:        return
11:    end if
12: end function
```

本研究では探索手順における、子ノードの選択確率 p_i に関して、以下に示す 3 つの確率設定手法を提案する。

3.1 提案手法 1：一様確率設定

提案手法 1 では、子ノードに対して一様に確率を設定する。現探索ノードにおける i 番目の子ノードの選択確率を以下に示す。

$$p(i) = 1/m. \tag{13}$$

3.2 提案手法 2 : 支持度 (support) の情報を加えた確率設定

提案手法 1 は、入力グラフの特徴を利用しないナイーブな手法である。本節では、入力グラフに対する支持度 (support) の情報を利用した手法を提案する。支持度とは、ある部分グラフが全入力グラフの内いくつのグラフで現れるかを表す値であり以下の式で定義される。

$$support(x) = \sum_{n=1}^{l} \mathbb{I}(x \subseteq G_n)(x \in \mathcal{T}). \tag{14}$$

探索木の性質上、支持度が大きい部分グラフのほうが小さいものに比べて複数の入力グラフ上での拡大グラフを考えることができるため、子孫空間が広いことを期待できる。子孫空間が狭いノードに多くの $flow$ を流す場合、探索が広がらず無駄になる可能性が大きい。したがって、本手法では支持度が大きい子ノードが選択されやすくなるよう確率を定める。i 番目の子ノードが表す部分グラフを x_i とすると、子ノードの選択確率を以下の式で示す。

$$p(i) = support(x_i) / \sum_{n=1}^{m} support(x_n). \tag{15}$$

3.3 提案手法 3 : bound,gain の情報を加えた確率設定

本節では、gain と bound の情報を利用した手法を提案する。gain、bound は、本稿第 2 章で定義したものを用いる。bound は子孫ノードで得ることのできる gain の上限値を表すため、bound の大きな子ノードへの探索を行うことで、より大きな gain の値を得ることが期待できる。また、gain の値は bound の値との探索空間上の距離を計る指標になることが期待できるため、gain と bound の凸結合の値を用いて子の選択確率を定めることで、より効率的な探索を行う。子ノードの選択確率を以下の式で示す。

$$\begin{aligned} p(i) &= val(x_i) / \sum_{n=1}^{m} val(x_n), \\ val(x_i) &= \lambda gain(x_i) + (1-\lambda) bound(x_i). \end{aligned} \tag{16}$$

λ は gain と bound の重みを表すパラメータであり、$0 \leq \lambda \leq 1$ を満たす。

3.4 探索空間の同型判定除外

既存手法で用いられる DFS コード木は gSpan[3] を用いて構築される木である。gSpan アルゴリズムでは、グラフ同型問題を解き、同型である場合には当アルゴリズムで定義される辞書順を用いて最小のものだけを木に採用する。当アルゴリズムにおいて辞書順最小のものは木全体に対して左側に出現しやすいため、DFS コード木は木全体の形が左寄りになるという特徴を持つ。ここで問題となるのが、探索木が左寄りである時、提案手法 1 のように一様に確率を設定する場合においても、同階層のノードへの探索確率に大きな差が生じてしまうということである。理解を簡単にするため、以下に図を示す。

図

本研究では、gSpan アルゴリズムでのグラフ同型判定を行わずに、辞書順最小でないノードも木に追加することで、木全体の偏りを緩和し、以上のような探索確率の差が軽減するよう試みる。提案手法 2、3 においても、同一の探索木を利用する。

4 実験&結果

提案手法を用いて構築されるモデルの評価を行うために、3 つの実験を行う。

データ名	グラフ数	平均ノード数	平均エッジ数	正例	負例
CPDB	684	25.2	25.6	341	343
Mutag	188	26.3	28.1	125	63

表 1: 使用したデータセット

4.1 データセット

本稿では、Saigo ら [1] によって用いられた 2 つのデータセットを使用した。表 1 に各データセットのグラフ数、平均ノード・エッジ数、正例と負例の数を示す。

4.2 実験 1：探索の精度、効率に関する実験

実験 1 では、CPDB と Mutag のデータに対して各提案手法がどれだけ精度よく探索できているか、どれだけ既存手法より効率化できているかを計る実験を行う。本稿では、主問題における目的関数 (2 章 (5) 式) の値を探索精度の指標に、モデル構築までに gain と bound の値を計算したノードの総数を探索ノード数とし、効率化の指標に用いる。

以下では、誤分類に対するコスト制御のパラメータ ν の値が 0.1、0.3、0.5 の 3 つの場合に対して実験を行う。

4.2.1 flow - 目的関数

$flow$ の数を変化させた時の目的関数の最終値の推移を調べる。$flow$ の刻み幅は、提案手法 1、2 に対しては 10 とする。提案手法 3 は、アルゴリズム上、子ノードの gain と bound を全て計算する必要があるため、提案手法 1，2 と比べて探索ノード数が多くなる。したがって、$flow$ の刻み幅は 1 とする。

4.2.2 結果

4.2.3 flow - 探索ノード数

$flow$ の数を変化させた時の探索ノード数の推移を調べる。$flow$ の刻み幅は、前述したものと同じ値とする。

4.2.4 結果

4.2.5 探索ノード数 - 目的関数

各提案手法の探索ノード数に対して、目的関数の最終値の推移を調べる。

4.2.6 結果

4.3 実験 2：λ に関する実験

実験 2 では、提案手法 3 における gain と bound の重みパラメータ λ の値を変化させた時の目的関数の最終値、探索ノード数の推移を調べる。$flow$ 数を 100 に固定し、λ の値を 0 から 1 まで 0.1 ずつ変動させる。これをデータに対し 10 fold cross varidation を行う。以上の実験を実験 1 と同様に、ν の値が 0.1、0.3、0.5 の場合に対して行う。

4.4 実験 3：乱数のブレに関する実験

提案手法では、乱数を使用しているため、結果にブレが生じる。したがって、実験 3 では各提案手法が乱数にどれだけ影響を受けるのか、そのブレを調べる。$flow$ 数を 100 に固定し、乱数に 10 個のシード値を与えた時の目的関数の最終値、探索ノード数の平均、分散値を調べる。以上の実験を実験 1 と同様に、ν の値が 0.1、0.3、0.5 の場合に対して行う。

謝辞

参考文献

[1] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, Vol. 75, No. 1, pp. 69–89, 2009.

[2] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, Vol. 46, No. 1-3, pp. 225–254, 2002.

[3] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 721–724, 2002.