3A2-1

大規模なL1正則化問題のための座標降下法を用いたスキーム

Scheme for Large-scale L1-regularized Problems using Coordinate Descent Methods

松島 慎 *1
Shin Matsushima

*1東京大学　情報理工学系研究科
Graduate School of Infomation Scinence and Technology, The University of Tokyo

L1 正則化を用いた経験誤差最小化問題を解くことによって、大量の特徴を持つデータから前もって特徴選択を行うことなく有用な学習器とその学習器に必要な特徴を同時に得る方法が機械学習の分野では広く利用されている。本発表では全ての特徴を展開した場合データが膨大になり使用可能なメモリ容量を超過してしまうような場合に、全ての特徴を展開したデータを持つことなく学習器の訓練と必要な属性の抽出を同時に行うスキームを提案し、その有用性を確認する。

1. はじめに

本稿では $\mathbf{w} \in \mathbb{R}^p$ をパラメータとする以下の関数を最小化することを考える：

$$P(\mathbf{w}) \triangleq \|\mathbf{w}\|_1 + C\sum_{i=1}^{n} \ell(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle, y_i). \qquad (1)$$

ここで $\ell : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ は微分可能な凸関数とし、$\mathbf{x}_i \in \mathcal{X}$ と $y_i \in \mathcal{Y}$ は学習に用いる入力と出力とする。また、$\phi : \mathcal{X} \to \mathbb{R}^p$ はデータの特徴を記述する関数である。この最小化問題は機械学習の分野で L1 正則化項付きの経験誤差最小化問題と呼ばれる一般的な枠組みで現れ、最小解を用いて得られる予測器は分類問題や回帰問題だけでなく、構造予測など複雑な出力を入力から予測する場合にも多く用いられる [7, 9]。

一般にサンプル数 n が増大するにつれ、ϕ として複雑かつ高次元な関数を用いても過学習が起こらず、結果として性能の高い予測器を得ることが可能になる。近年ニューラルネットを用いて表現されるような複雑な予測器で高精度な予測が可能になったのも、利用可能なサンプル数が増大したことが一つの要因になっている。このことからも線形予測器においても高精度な予測器を得るためには、ϕ として複雑かつ高次元な関数を用いることが有効であると考えられる。

(1) の最小解を求める際、$\Phi \triangleq \{\phi_j(\mathbf{x}_i)\}_{1\le i\le n, 1\le j\le p}$ の計算と実際の求解の段階は分離されている。そのため非常に大きな数の特徴を考えることで Φ のデータ量が計算機のメモリ容量を超す場合に、最適化を効率的に行うことができない問題がある。すなわち、典型的には勾配を求める場合などに Φ の値を読み込む際に、低次の記憶領域にアクセスしなければならず、実際の計算時間が大きくなってしまう。

このような問題は Yu et al.[12] によって最初に指摘され、ここではメモリ容量より大きなデータを用いて正則化付き経験誤差最小化問題を解く最初のスキームとして、ブロック最小化スキームが提案されている。ブロック最小化スキームや Matsushima et al. の Dual Cached Loops [4] は確率的勾配法と組み合わせることによって (1) の最小化問題を解くことが可能である。

一方で、L1 正則化を用いて予測器を学習する際は、予測に必要ない特徴が多く存在し最適解として疎な解を得られることが期待される。すなわち、出力の予測のために必要な特徴は特徴関数のうち一部の関数のみであり、L1 正則化の影響で必要のない特徴に対応するパラメータが零化されることが期待される。そのような場合、不必要と推測される特徴を無視して最適化を行うことで経験的に学習の効率を上げる方法が提案されている [1]。このような方法を利用すれば、必要な特徴のみを展開しメモリに載せることで、データがメモリに載らない場合に生ずる問題を回避することができると考えられる。従来の確率的勾配法などに基づく手法では不必要な特徴を展開させる必要があるため、(1) の最小化問題のためにはこのような方針が有効であると考えられる。

本発表では、上述した方針に基づき Φ がメモリ容量より大きいような場合に最適化問題 (1) を効率的に解くためのスキームを提案する。本スキームでは二種類の異なるアルゴリズムを非同期的に動作させることで特徴の選択的抽出とパラメータの最適化を同時に行う。実験においては、32GB の記憶容量を用いて、9TB から 36TB に対応する大規模なデータセットから有効な学習が行えたことを報告する。

2. L1 正則化問題のための座標降下法

座標降下法は一般的な凸関数を最小化するための最適化法として知られており、最近では機械学習の文脈で現れる最適化問題を効率的に解くことができることを多くの研究が報告している [2, 5, 11]。本節では、(1) の最小化問題における座標降下法の枠組みを説明する。

一般的な座標降下法は各反復でパラメータの一部の変数にだけ着目することで得られる部分問題を解くことで、対応するパラメータのみを更新する。ここでは、一変数のみに着目し、時刻 t におけるパラメータ $\mathbf{w}^t$ に対する更新式は $\mathbf{w}^{t+1} = \mathbf{w}^t + \delta_j^t \mathbf{e}_j$ とする。$\mathbf{e}_j$ は j 番目の標準基底である。また、

$$\delta_j^t \triangleq \operatorname*{argmin}_{\delta} \left\|\mathbf{w}^t + \delta \mathbf{e}_j\right\|_1 + C\left(\nabla_j L(\mathbf{w}^t)\delta + \frac{1}{2}\nabla_{jj} L(\mathbf{w}^t)\delta^2\right)$$

と定義する。ここで、

$$\nabla_j L(\mathbf{w}) \triangleq \frac{\partial}{\partial w_j} \sum_{i=1}^{n} \ell(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle, y_i)$$

$$\nabla_{jj} L(\mathbf{w}) \triangleq \frac{\partial^2}{\partial {w_j}^2} \sum_{i=1}^{n} \ell(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle, y_i)$$

である。また、より安定的に解を求めるために、次の条件を満たすまで $0 < \beta \leq 1$ を減少させることで更新幅を縮小する方法がよく用いられる [10, 13]：

$$P(\mathbf{w}) - P(\mathbf{w} + \beta\delta\mathbf{e}_j) \geq \sigma\beta \left(C\nabla_j L(\mathbf{w}^t)\delta + |w_j + \delta| - |w_j| \right), \quad (2)$$

ここで $\sigma > 0$ は適当に定められる定数である。まず、$\beta = 1$ かつ $\delta = \delta_j^t$ として条件 (2) を満たすかどうか調べ、これを満たすまで β を指数的に減少させる。得られた β を β^t として最終的な更新は次のように表せる：

$$w_j^{t+1} = w_j^t + \beta^t \delta^t. \quad (3)$$

これらの更新に必要な $\nabla_j L(\mathbf{w})$ および $\nabla_{jj} L(\mathbf{w})$ の計算は次の補助パラメータ $u_i^t \triangleq \langle \mathbf{w}^t, \phi(\mathbf{x}_i) \rangle$ を用いることで $\mathrm{O}(|\Omega_j|)$ の計算時間で計算可能である。ここで $\Omega_j \triangleq \{i | \phi_j(\mathbf{x}_i) \neq 0\}$ である。この補助パラメータも同様に $\mathrm{O}(|\Omega_j|)$ の計算時間で以下の等式を用いて更新可能である：

$$u_i^{t+1} = u_i^t + (w_j^{t+1} - w_j^t) x_{ij}.$$

座標降下法が一回のパラメータ更新で必要な値は u_i, ϕ_j と w_j のみであり、Φ 全体の情報は必要ない。そのため、データが全てメモリ上にない場合でも効率的に更新を行うことができる。一般的な場合には (1) を解くためのより収束速度の速いアルゴリズムが提案されている [14] が、座標降下法はデータがメモリ容量を超える場合の問題により適していると考えられる。

3. 提案スキーム

本節では大量の特徴を有するデータセットを扱うための提案スキームである特徴キャッシュループ（Feature Cached Loops, FCL）について説明する。本スキームは任意の基底関数について学習することができるスキームであるが、基底関数が持つ構造を用いて効率的な学習を行うことも可能である。本スキームでは書き込みスレッドと訓練スレッドと呼ばれる二つのスレッドがパラメータ $\mathbf{w}$、補助パラメータ $\mathbf{u}$ と特徴キャッシュ Φ_J を共有し、非同期的に動作する。書き込みスレッドはデータから特徴 ϕ_j を読み込み、これを特徴キャッシュに追加する。ただし、抽出された特徴が現在の解を改良することができなければ、特徴キャッシュに追加し訓練スレッドと共有することなく、展開された特徴は破棄される。すなわち特徴を破棄するための条件は

$$-1 < C\nabla_j L(\mathbf{w}^t) < 1. \quad (4)$$

と記述することができる。この値は前述のように補助パラメータ $\mathbf{u}^t$ を用いて簡単に計算することができる。もし特徴キャッシュの容量を超過するようであれば、無作為にキャッシュ内の特徴を破棄し、解放された領域に特徴を追加する。書き込みスレッドの擬似アルゴリズムを Algorithm 1 に記述する。

Algorithm 1 書き込みスレッドのアルゴリズム

```
J ← ∅, Φ_J ← ∅
訓練スレッドと J, Φ_J, w, u を共有する
while 訓練スレッドが動作中 do
    無作為に j ∈ {1, ..., p} を選択する
    φ_j = {φ_j(x_i)} を展開する
    φ_j および u を用いて ∇_j L(w) を計算する
    if C|∇_j L(w)| > 1 then
        Φ_J ← Φ_J ∪ {φ_j}
        J ← J ∪ {j}
    end if
    while |J| > 特徴キャッシュの容量 do
        無作為に j' ∈ J を選択する
        J ← J \ {j'}
        Φ_J ← Φ_J \ {φ_j'}
    end while
end while
// no output
```

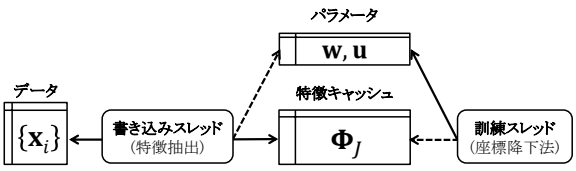

図 1: 特徴キャッシュループ（FCL）の模式図。それぞれのスレッドが共有メモリにアクセスするが、点線は主に読み込みのみ実線は書き込みを含むアクセスを示す。

一方で、訓練スレッドは特徴キャッシュ内に格納されている列に対応する座標を無作為に一つ選び、座標降下法を用いてパラメータ更新を行う。選んだ座標に関して (4) かつ $w_j = 0$ が成り立つ場合、対応する列は特徴キャッシュから破棄することで座標降下法の効率を上げることを目指す。この条件は前節で説明したヒューリスティクスに比べより多くの列を不必要とみなすため、実際には必要だった列を破棄する機会がより多くなると考えられる。しかし書き込みスレッドが同時に各列の必要性を確認しているため、より積極的に列を除外することが全体の効率を上げると考えられる。訓練スレッドの擬似アルゴリズムを Algorithm 2 に記述する。

一般に、書き込みスレッドは特徴を展開することに大部分の時間を費やす一方で、訓練スレッドはパラメータの更新に大部分の時間を費やすため、特徴キャッシュへのアクセスにおける衝突やそれに伴うスレッドの待機はほとんど生じないと考えられる。特徴キャッシュに関しては訓練スレッドに関しては非同期的な座標降下法 [3, 6] を用いて複数のスレッドを利用することができる。また書き込みスレッドは自然に同時に複数のスレッドを利用することができる。提案スキームの全体の模式図を図 1 に示す。

表 1: 実験で用いたデータセットの属性

dataset	# of instances	# of features	# of non-zero elements	sparsity (%)	data size (GB in text file)
splice($d = 8$)	50,000,000	41,875,000	875.6×10^9	0.410	$< 9,000$
(sampled)	50,000,000	200,000	4.179×10^9	0.418	44.44
splice($d = 10$)	50,000,000	1,031,250,000	3.38×10^{12}	0.066	$< 36,000$

Algorithm 2 訓練スレッドのアルゴリズム

$\mathbf{w} \leftarrow \mathbf{0}$, $\mathbf{u} \leftarrow \mathbf{1}, \sigma \leftarrow 0.01$
書き込みスレッドと J, Φ_J, $\mathbf{w}$, $\mathbf{u}$ を共有する
while not converged **do**
  無作為に $j \in J$ を選択する
  Φ_J から ϕ_j を読み込む
  ϕ_j および $\mathbf{u}$ を用いて $\nabla_j L(\mathbf{w}), \nabla_{jj} L(\mathbf{w})$ を計算する
  if $C|\nabla_j L(\mathbf{w})| < 1$ **then**
    $J \leftarrow J \setminus \{j\}$
    $\Phi_J \leftarrow \Phi_J \setminus \{\phi_j\}$
  end if
  $\delta \leftarrow \operatorname{argmin} |w_j + \delta| + \nabla_j L(\mathbf{w})\delta + \frac{1}{2}\nabla_{jj} L(\mathbf{w})\delta^2$
  $\beta \leftarrow 1$
  while (2) を満たさない **do**
    $\beta \leftarrow 0.9\beta$
  end while
  $w_j \leftarrow w_j + \delta\beta$
  $u_i \leftarrow u_i + \delta\beta\phi_j(\mathbf{x}_i)$ for each $i \in \Omega_j$
end while
output $\mathbf{w}$

4. 実験結果

本節では、長さ 141 の {A,T,C,G} からなる DNA 配列からその配列がスプライス部位を含むか否かの識別を行う問題を L1 ロジスティック回帰を用いて学習するタスク [8][*1] を通じて、大規模な特徴関数を用いた学習が可能であることを計算機実験により示す。

本実験では DNA 配列に対して以下の特徴を考える。まず、次数 d を定め、長さ d の {A,T,C,G,?} からなる文字列が正規表現としてデータの $141 - k - d$ 番目から $141 - d$ 番目までにマッチするならば 1、そうでなければ 0 となる特徴を考える。'?' から始まる文字列を除外したすべての長さ d の部分文字列と $1 \leq k \leq 142 - d$ の組に関し特徴が定まるため、全体で $(142 - d) \cdot 4 \cdot 5^{d-1}$ 個の特徴があり、各データは $(142 - d) \cdot 2^{d-1}$ 個の非零要素を含む。本実験では $d = 8, 10$ を用いた。この場合の具体的なデータセットの各属性の値を表 1 に示す。

学習時間と目的関数の値および AUPRC (Area Under Precision-Recall Curve) の関係を調べた。実装は C++ を用いて行い、全ての計算は Intel Xeon CPU X5690 (3.47 GHz) を用いた。本実験においては特徴キャッシュの容量を 32GB とし、書き込みスレッドを同時に 5 つ動作させた。比較として、45GB 分の特徴を無作為に抽出したデータセットを用意し、これに関して L1 ロジスティック回帰を行った場合の目的関数の値および AUPRC の

*1 http://sonnenburgs.de/soeren/item/coffin/

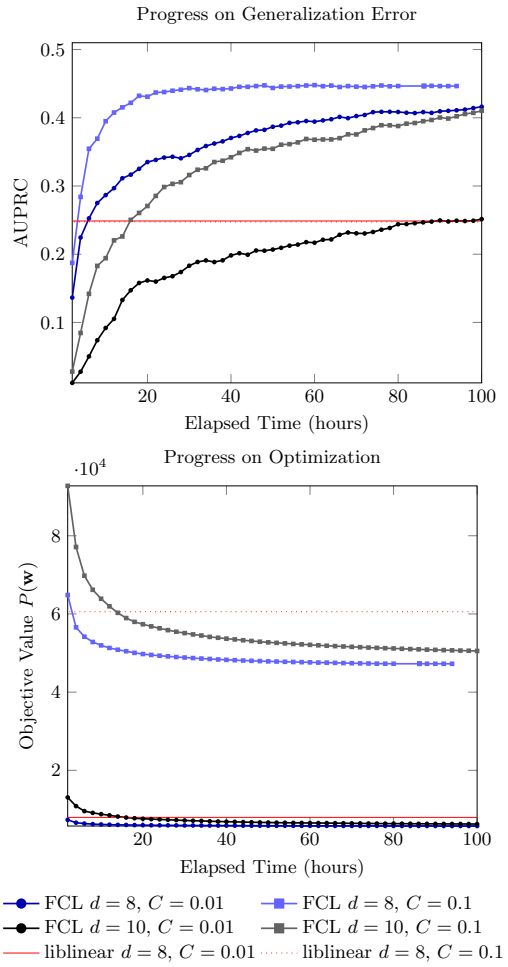

図 2: スプライス部位認識タスクにおける AUPRC（左図）と目的関数値（右図）の推移

値を図 2 に示した。全ての場合で無作為に特徴を抽出した場合よりもより最適解に近い解が得られ、関数値もほぼ最適に近い値を達成していると考えられる。AUPRC も 45GB のデータのみを用いるより高い値が得られた。$d = 10$ の場合では 100 時間を超えてもまだ最適化は終わっていないが　AUPRC の値としては高い値を取り得ると考えられる。より大規模なデータを用いた学習を単一の計算機で行うためにはより効率的なスキームが望まれる。

5. 結論

本発表では大量のデータとそれに伴う大量の特徴を用いた L1 正則化付き経験誤差最小化問題を解くスキームを提案した。提案スキームは非同期的に同時に各データ

の特徴量の展開とパラメータの最適化を行うことで、必要そうな特徴のみに関して効率的に学習を行うことができることを実験的に示した。実験では組み合わせ的な特徴を用いたが他にも様々な特徴を用いることができる。さらに特徴空間の構造を利用することにより、より効率的な特徴抽出が可能であると考えられる。

謝辞

本研究は JSPS 科研費 JP26730114 およびマイクロソフトリサーチ CORE 連携研究プログラムの助成を受けたものである。

参考文献

[1] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[2] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S.-S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of International Conference on Machine Learning*, pages 408–415, 2008.

[3] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015.

[4] S. Matsushima, S.V.N. Vishwanathan, and A. J. Smola. Linear support vector machines via dual cached loops. In *Proceedings of Knowledge Discovery and Data Mining*, pages 177–185, 2012.

[5] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[6] Z. Peng, Y. Xu, M. Yan, and W. Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):A2851–A2879, 2016.

[7] I. Rish and G. Grabarnik. *Sparse Modeling: Theory, Algorithms, and Applications*. CRC Press, Inc., 2014.

[8] S. Sonnenburg and V. Franc. COFFIN: A computational framework for linear SVMs. In *Proceedings of International Conference on Machine Learning*, pages 999–1006, 2010.

[9] R. Tibshirani. The lasso method for variable selection in the cox model. In *Statistics in Medicine*, pages 385–395, 1997.

[10] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[11] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[12] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *Proceedings of Knowledge Discovery and Data Mining*, pages 833–842, 2010.

[13] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.

[14] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. An improved glmnet for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.