F-016

協調フィルタリングにおける制約ボルツマンマシンの応用

Application of Restricted Boltzmann Machines to Collaborative Filtering

豊原 優† 藤森 進‡ 太原 育夫‡
Suguru Toyohara Susumu Fujimori Ikuo Tahara

1. はじめに

現在，広告業界では「アドエクスチェンジ」によりインターネット上の広告枠の売買取引システムが確立されている．ホームページに設置された広告枠のほとんどがマーケット上で取引されており，広告の価値を大きく左右する要因として，広告を見るターゲットの適切な設定が考えられる．適切にユーザを見つけ出すシステムの性能が高いほど広告の費用対効果は高くなると予想される．

適切に商品とユーザをつなげるシステムは現在様々な形で研究が進められており，こうしたシステムは「推薦システム(Recommend System)」と呼ばれている．その一つに協調フィルタリング(Collaborative Filtering; CF)がある．

協調フィルタリングとは，N 人のユーザと M 種類の商品との評価マトリクスを用いて，推薦を行うユーザと選択傾向が近いと思われるユーザ群を抽出し，そのユーザ群から当該ユーザが未だ購入していないアイテムを推薦することで，クチコミのような効果を当該ユーザにもたらすシステムである [1]．実装は容易であるが，予測が出来る商品が限られてしまうという問題点がある．従来のシステムではユーザ間の相関係数を求めるため，他のユーザと共通する商品を評価していない場合予測値が算出できない．

本論文で用いる手法は 2007 年に提案された手法[2]を用いており，この手法の特徴は制約ボルツマンマシン(以下，RBM)を用いて CF では予測値を算出できないユーザに対しても算出が可能であり，CF よりも精度が高いという点にある．しかし，理論が複雑で一般的に学習にかかる計算量が多く，精度を重視して処理速度についてはそれほど考慮していないと思われる．そこで，本研究では RBM を Python 上で実装し CF に応用した場合，精度と処理時間を検証し，現実のアドネットワークシステムに対して速度面で応用可能であるかを検討した．

2. RBM の協調フィルタリングへの応用

2.1 RBM

RBM は，入力となる可視層(visible layer)と隠れ層(hidden layer)の二層構造のニューラルネットワーク構造を持ち，可視ノード間，隠れノード間には接続がないボルツマンマシンである．

RBM における入力と出力は可視層のバイナリ値であり，入力されたバイナリ値は重みとバイアス項を加算し活性化関数によって隠れ層の確率表現が生まれる．重みベクトル W，可視層バイナリ値ベクトルを$\mathbf{v}$，隠れ層バイアス項を$\mathbf{b}$，シグモイド関数をσとすると，隠れ状態ベクトル $\mathbf{h}$ は

$$\mathbf{P}(\mathbf{h}|\mathbf{v}) = \sigma(\mathbf{b} + \mathbf{v} \cdot \boldsymbol{W})$$

と確率表現できる．ここからサンプリングを行い，バイナリ化とすると隠れ層のバイナリベクトル$\mathbf{h}$となり，同じ重み$\boldsymbol{W}$，可視層バイアス項$\mathbf{c}$，を用いて可視層状態ベクトル $\mathbf{v}$ は

$$\mathbf{P}(\mathbf{v}|\mathbf{h}) = \sigma(\mathbf{c} + \boldsymbol{h} \cdot \boldsymbol{W^T})$$

と逆方向の伝搬により確率表現できる．同様にサンプリングを行い，元のデータと比較することにより学習を行う．

RBM における学習とは，この入力から出力した確率からサンプリングしたバイナリ値が元の入力と同じとなるように，重みとバイアス項を調整することである．

2.2 協調フィルタリングとしての RBM

この節では，RBM をどのように協調フィルタリングへ応用するかについて説明する．

バイナリ値を入力とする RBM において商品と評価を RBM に入力するため，次のように RBM を拡張して考える．

図 1 のように，可視層を最大評価値 K の分だけ増やして考える．ここで注意すべきは，RBM の学習はユーザごとに学習させ，RBM はそのユーザが持つ特徴を再現するのが目的だということである．可視層を K 倍に増やすことで，ユーザのアイテム評価値をバイナリ値で表現することが可能となる．また，個々で学習した重みに関して，全 RBM における隠れノードの数を固定にすることで，隠れノードと全てのアイテム（可視ノード）との間に接続する重みが完全に特定することが出来るため，全 RBM 間で共有して使い回すことが可能である．このことにより，ユーザ自身を学習した RBM の隠れノードに対し，全 RBM 間で学習した重みを使い回すことで，ほぼ全てアイテムの評価予測値を算出することが可能となる．

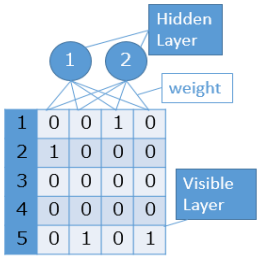

図 1 拡張した RBM のイメージ

しかし，この方法で学習させようとすると重みが三成分の行列データを持つことになり，行列演算をそのまま用いることが困難となる．For 文による繰り返しの演算は可能だが，Python 上で実装することを考慮した場合，その演算に掛かる処理は膨大なものとなり，現実的ではない．また，推薦システムがアドネットワーク上で利用可能とされる処理時間である 50 ミリ秒以内[3]を達成するには，行列演算処理を行うのが必要不可欠である．

行列演算を行うためには，行列を k 成分で輪切りにして演算させる方法が適用できる．伝搬の計算である $\mathbf{P}(\mathbf{h}_j = \mathbf{1}|\mathbf{v})$は以下のように表現できる．

$$\mathbf{P}(\mathbf{h}_j = \mathbf{1}|\mathbf{v}) = \sigma\left(\mathbf{b} + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k w_{ij}^k\right)$$

そして，行列を用いれば以下のように表現できる．

$$\mathbf{u}^k = \mathbf{v}^k \cdot \boldsymbol{w}^k$$

† 東京理科大学大学院工学研究科経営工学専攻
‡ 東京理科大学工学部第二部経営工学科

$$P(\mathbf{h}|\mathbf{v}) = \sigma\left(\mathbf{b} + \sum_{k=1}^{K} \mathbf{u}^k\right)$$

逆方向への伝搬は出力先が多層であることを考慮して

$$P(v_i^k|\mathbf{h}) = \frac{\exp\left(c^k + \sum_{j=1}^{F} h_j w_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(c^l + \sum_{j=1}^{F} h_j w_{ij}^l\right)}$$

であるから

$$\mathbf{u}^k = \exp\left(\mathbf{h} \cdot \left(w^k\right)^T + c^k\right)$$

$$U = \begin{pmatrix} u^1 \\ u^2 \\ \vdots \\ u^k \end{pmatrix} = \begin{pmatrix} u_1^1 & \cdots & u_m^1 \\ \vdots & \ddots & \vdots \\ u_1^K & \cdots & u_m^K \end{pmatrix}$$

となり，この行列 $\boldsymbol{U}$について，列ごとに列ごとの総和で割ることによって可視状態マトリクス $\boldsymbol{V}$が得られる．

$$P(\mathbf{v}|\mathbf{h}) = \begin{pmatrix} \frac{u_1^1}{\sum_{l=1}^{K} u_1^l} & \cdots & \frac{u_m^1}{\sum_{l=1}^{K} u_m^l} \\ \vdots & \ddots & \vdots \\ \frac{u_1^K}{\sum_{l=1}^{K} u_1^l} & \cdots & \frac{u_m^K}{\sum_{l=1}^{K} u_m^l} \end{pmatrix}$$

以上により，伝搬および逆伝搬の式を用いれば，CD 法の処理が速くなり，簡単に短時間で勾配計算をすることが出来ると言える．

3. 実験

3.1 実験条件

表 1　実験機スペック

項目	使用環境
使用OS	Windows10
使用言語	Python3.5.3
CPU	Intel®Core™i7-4729
CPUクロック数	3.60GH z
RAM領域	32GB

表 2　サンプルデータ詳細

	MovieLens-100K
総ユーザ数	943
総映画数	1682
評価数	100000
評価密度	6.304%

検証を行った際のマシンスペックは表 1 の通りである．なお，RBM の初期設定値はそれぞれ，隠れノード数は 100 固定，重みは平均 0 標準偏差 0.01 の正規分布乱数，$p = 1/K$として $b^k = \ln(p/(1-p))$の定数で設定している[4]．

サンプルデータには，Movie Lens[5]にて配布されている MovieLens-100K データセットを用いた(表 2).

このデータセットを，ランダムに教師データと正解データに 8：2 で分割し，8 割の教師データを元にすべての予測値を計算し，2 割の正解データと照らし合わせることで精度を計算する．また，交差検証法を用いてすべてのデータセットを用いて比較することで，正確なモデル分析も行う．精度の計算は RMSE 値を用いる．

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}$$

3.2 結果

表 3　各手法の精度

	data1	data2	data3	data4	data5	Avarage
Random-Accurate	1.9625	2.0364	2.1329	1.9041	1.9954	2.0062
CF-Accurate	1.0421	1.0444	1.0488	1.0544	1.0480	1.0475
RBM-Accurate	0.8961	0.8709	0.9835	0.9503	0.9569	0.9315
CF-cover rate	81.115%	82.450%	82.125%	82.515%	82.370%	82.12%
RBM-cover rate	97.681%	98.335%	98.098%	98.098%	98.335%	98.11%

精度比較(表 3)に関しては，すべてのデータセットに対して RBM が CF の予測精度を上回っていることが分かる．

表 4　処理時間 (単位:秒)

	get similarity		prediction	prediction/user	append
CF	310.1194636		549.2841	0.582485836	0.682168
	initialize	learning	prediction	prediction/user	append
RBM	0.806788	5.687843	4.552606	0.004827791	0.00672

処理時間の結果を表 4 に示す．CF における get-similarity は全ユーザの総当たりの類似度を計算するのに必要な時間であり，learning は全 RBM の総学習時間を表し，prediction は予測に掛かった時間を表している．append は 1 つ評価値を加えたときに再構築に必要な時間を表している．RBM による手法は，Python においてはどの項目よりも CF より優れていると言える．構築時間は 50 倍速く，予測値と加算処理は 100 倍速いと分かる．また，データの規模にもよるが，1 人のユーザに対する予測値計算は 50ms を十分に下回っていることから，広告取引システムに組み込むことが出来るほど速いことが分かる．

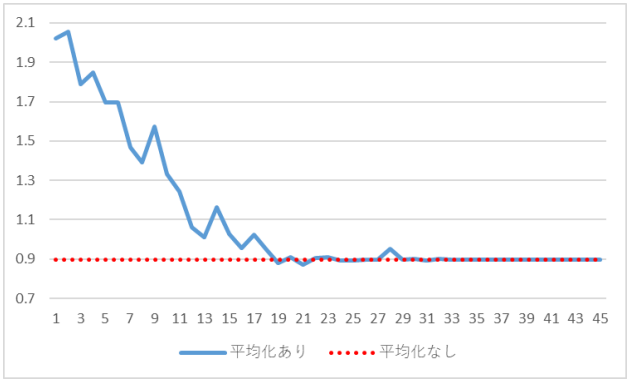

図 2　平均化ありと平均化なしとの収束の違い

また，学習によって計算された重みを，全アイテム数で割った平均化処理し，共通重みを更新した場合と，平均化処理を行わない場合とで比較したところ，図 2（縦軸は RMSE，横軸は学習回数）に示すように，どちらも同じ値に学習が収束し，平均化処理を行わなかった場合は一度の学習でパラメータが収束することが分かった．

4. おわりに

本研究では，Python にて RBM を用いた CF の応用とその検証を行い，Python を用いた速い処理速度と高い精度を持つ推薦システムを開発した．これによってより簡単に精度の良い推薦システムをサーバに組み込むことが可能となった．今後の課題としては隠れ層の最適化などが挙げられる．

参考文献

[1] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich (田中克己,角谷和俊(監訳)), 情報推薦システム入門 理論と実践, 共立出版, 2012.
[2] R. Salakhutdinov, A. Mnih , G. Hinton,“Restricted Boltzmann Machines for Collaborative Filtering,” the 24th International Conference on Machine Learning, 2007.
[3] DSP, SSP の仕組みと特徴-デジタルマーケティングラボ http://dmlab.jp/adtech/dsp.html (2017 年 6 月 28 日 閲覧)
[4] G. Hinton,“A Practical Guide to Training Restricted Boltzmann Machines”, Department of Computer Science University of Toront, 2010. (http://learning.cs.tronto.edu)
[5] Movie Lens – Group lens http://grouplens.org/datasets/movielens/ (2017 年 6 月 28 日閲覧)