数値属性の最適結合ルールを発見する効率的アルゴリズム

福田剛志†

巨大データベースの中に潜在する数値属性の結合ルール（*association rule*）を発見する問題について述べる．たとえば，銀行の顧客のうち，預金残高がある区間 I に入る顧客は，カードローンを利用する確率が高いとする．この知識は (預金残高 $\in I$) $\Rightarrow$ (カードローン利用 $= yes$) という結合ルールとして表現できる．このようなルールは，区間 I が預金残高とカードローン利用の間の特別な性質を持つ場合についてのみ，面白いルールといえる．その性質とは，I に含まれる顧客の数（サポート）が十分に大きく，条件を満たす確率（確信度）が十分に高いことである．我々の目的は，そのような適切な区間を自動的に発見するシステムを実現することである．我々は主に2種類の最適な区間について考える: 1つは確信度がある閾値以上で，サポートが最大となる区間，もう1つはサポートがある閾値以上で，確信度が最大となる区間である．この論文ではこの問題を線形時間で解くアルゴリズムを提示し，その実装を評価して理論的計算量が確かに現実にも達成できたことを述べる．

Efficient Algorithms for Finding Optimized Association Rules for Numeric Attributes

TAKESHI FUKUDA†

Given a huge database, we address the problem of finding association rules for numeric attributes, such as (*Balance* $\in I$) $\Rightarrow$ (*CardLoan* $= yes$), which implies that bank customers whose balances fall in a range I are likely to use card loan with high probability. The above rule is interesting only if the range I has some special feature with respect to the interrelation between *Balance* and *CardLoan*. It is required that the number of customers whose balances are contained in I (*support*) is sufficient, and also that the probability of the condition being met (*confidence*) should be much higher than the average. Our goal is to realize a system that finds such appropriate ranges automatically. We mainly focus on computing two optimized ranges. In this paper, we present novel algorithms that compute the optimized ranges in linear time. Tests show that our implementation is fast not only in theory but also in practice.

1. はじめに

近年，バーコードリーダ，OCR，キャッシュディスペンサ，携帯端末などのデータ入力技術の進歩普及にともない，小売業者や金融業者が営業の現場で販売データや顧客データを収集することが可能になってきた．さらに，大容量のディスクの低価格化で，そのようなデータを大量に蓄えることが可能になってきた．このような背景の下に，日常収集された巨大なデータから，事業戦略に役立つ未知の知識を抽出するデータマイニング，あるいは，知識発見の技術が大変注目されている．しかし，現在のデータベースシステムはこの目的のための基本的な手段としては不十分であり，データ工学およびAIの研究者の間で，役に立つルールを効率的に発見する方法がさかんに研究されている[1)~5),7)~16)]．

1.1 結合ルール

あるデータベースのリレーションが与えられたとき，もしあるタプルが条件 C_1 を満たすならば，そのタプルは条件 C_2 をある確率（**確信度** *confidence* と呼ぶ）で満たすといった，条件間の関係を考えることができる．我々はこれを C_1, C_2 の間の**結合ルール**（*association rule*，あるいは単にルール）と呼び，$C_1 \Rightarrow C_2$ のように表記する．

確信度が100%に近いルールを発見するための方法としてKID3アルゴリズム[12)]が提案されている．科学技術分野では，重要なルールは厳密であることが多いと考えられるのでKID3は有効である．しかし，顧客データなどの商業的なデータは現実世界のさまざまな予期できない条件に左右されるため，重要なルールといえどもその確信度は小さくなる．

† 日本アイ・ビー・エム（株）東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.

したがって，商業的なデータベースを取り扱うためには，厳密なルールだけでなく，ある程度以上の確信度を持つ広い範囲のルールを考えなければならない．Agrawal らは文献 3) で，このようなルールを求める方法について考察している．彼らはそこで，2 値属性の論理積を条件とし，この条件に関してユーザが与える閾値以上のサポートを持つすべてのルールを求める効率的なアルゴリズムを示した．さらにこのアルゴリズムを小売トランザクションのデータに適用して，次のような商品間の関連購買現象を表すルールを発見するために用いた：

(ピザ＝*yes*) ∧ (コーラ＝*yes*) ⇒ (ポテト＝*yes*)

このアルゴリズムの改良が文献 4), 11) に報告されている．

1.2 最適結合ルール

現実のデータベースには 2 値属性だけでなく，年齢や預金残高といったような，一般の数値属性が存在し，数値属性の結合ルールを発見することも重要な問題である．本論文では，次のような形式の結合ルールを発見することに注目する：

(預金残高 ∈ $[v_1, v_2]$) ⇒ (カードローン = *yes*)

このルールは預金残高が v_1 から v_2 の間であるような顧客は，ある高い確信度でカードローンを利用しているということを表している．もし適当な区間があらかじめ与えられていれば，その確信度を求めることは簡単である．しかし現実には，確信度の高いルールを作るような区間自体を求めることが要求される．区間のとり方は自由度が大きく，ごくわずかなデータしか含まないが，高い確信度を与えるような区間をいくつも考えることができる．したがって，いくら確信度が高くても，あまり小さな区間では意味のあるルールとはいえない．

ある区間に入るデータ数が全体に占める割合を**サポート**（*support*）と呼ぶ．我々が求めたいルールは，サポートが十分大きく，しかも高い確信度を持つルールである．

特に，我々は与えられた閾値以上の確信度を持つルールのうち，サポートを最大とするルール（**最適サポートルール**，*optimized support rule* と呼ぶ）を求めることにする．このような区間は，閾値より小さくない確信度を持つ，最大のデータの集まりをとらえている．

また，最適サポートルールと双対の，**最適確信度ルール**（*optimized confidence rule*），すなわち与えられた閾値以上のサポートを持つルールのうち最大の確信度を持つルールも重要である．たとえば顧客 10% 以上を含む区間のうち，最も高い確率でカードローンを利用する区間を知ることができる．限られた予算内でダイレクトメールを発送し，カードローンの利用を促進したいならば，このようなルールは有効な顧客層を教えてくれる重要な情報となる．

2. 定　　義

2.1 結合ルール

データベースの任意のリレーションを R とする．R 中のタプルに関する条件を表すために，まず次の基本的な条件を用いる．ある 2 値属性 A に対して，$A = yes$, $A = no$ は基本的な条件である．また，ある数値属性 B に対して，$B = v$, $B \in [v_1, v_2]$ は基本的な条件である．さらに，これらの基本的条件の論理和積も条件となる．

たとえば，小売業の販売トランザクションデータにおけるリレーションを考えよう．その 1 つのタプルは 1 回の販売データを表す．属性は，ピザ，コーラなどの商品で，それが売れたか，売れていないかを *yes*, *no* で表す 2 値属性である．(ピザ = *yes*) ∧ (コーラ = *yes*) はこのリレーションに対する条件であり，この条件を満たすタプルは，ある顧客がピザとコーラを同時に購入したことを示す．

もう 1 つ，銀行の顧客データにおけるリレーションを例にあげよう．1 つのタプルは，ある顧客の預金残高と，各種のサービス（カードローン，公共料金の自動引き落としなど）を受けているかどうかであるとする．(預金残高 ∈ [15821, 26264]) ∧ (カードローン利用 = *yes*) はこのリレーションに対する条件となる．

条件 C に対するサポートは，条件 C を満たすタプルの全体に対する割合で，$sup(C)$ と表記する．先の小売業の例では，sup(ピザ) = 10% なら，10% の顧客がピザを買っていることになる．

あるリレーションに対する 2 つの条件を C_1, C_2 として，結合ルールは，$C_1 \Rightarrow C_2$ と表現される．ルール $C_1 \Rightarrow C_2$ の確信度は $sup(C_1 \wedge C_2)/sup(C_1)$ で定義され，$conf(C_1 \Rightarrow C_2)$ と表記する．先述の銀行の例では，ルール (預金残高 ∈ [15821, 26264]) ⇒ (カードローン利用 = *yes*) の確信度が 50% なら，預金残高が [15821, 26264] の範囲にある顧客の 50% がカードローンを利用していることになる．

2.2 最適化結合ルール

本論文では，$(A \in [v_1, v_2]) \Rightarrow C$ という形のルールに注目する．確信度が与えられた閾値 *minconf* より小さくないルールのうち，$sup(A \in [v_1, v_2])$ を最大とするルールを**最適サポートルール**と呼ぶ．また，

$sup(A \in [v_1, v_2])$ が与えられた閾値 *minsup* より小さくないルールのうち，確信度が最大であるものを**最適確信度ルール** と呼ぶ．

2.3 バケット

与えられたリレーション R のあるタプルを t とし，そのタプルの数値属性 A の値を $t[A]$ と表記することにする．属性 A の定義域を，次のような交わりのない**バケット**（*bucket*）の列

$B_1, B_2, \ldots B_M$

$(B_i = [x_i, y_i], x_i \leq y_i < x_{i+1})$

に分割し，すべてのタプルの属性 A の値が必ずどれかのバケットに入るようにする．このように分割すると，任意のタプル $t \in R$ に対して，$t[A]$ を含むあるバケット B_j がただ1つ存在することになる．さらに，$B_i = [x, x]$ であるようなバケット B_i を最小粒度のバケットと呼ぶ．

たとえば，属性 A が年齢を表しているとすると，A の定義域は正の整数で120以下であろう．したがって，121個の最小粒度バケット $[i, i]$ $(i = 0, 1, \ldots 120)$ を作ることができる．別の例として，A を銀行の数百万人の顧客各々の預金残高とすると，その定義域は0から 10^{10} まで及ぶかもしれず，結果として最小粒度のバケット数は（顧客の総数である）数百万に達することもありうる．

このようにバケットを用意しておくと，連続するバケット $B_s, B_{s+1}, \ldots B_t$ をつないで，区間 $[x_s, y_t]$ を作ることができる．すべてのバケットが最小粒度であれば，ある連続のバケットの連結が，最適結合ルールの区間となる．上に述べたように最小粒度バケットの数が巨大な場合は，最小粒度のバケットでなくても十分大きい数（たとえば数千）のバケットを考えれば，それらを連結することで，最適結合ルールの区間の実用上十分な近似を得ることができる．

さて，集合 $\{t \in R \mid t[A] \in B_i\}$ に入るタプル数を B_i の大きさ (*size*) と呼び u_i と表す．また，集合 $\{t \in R \mid t[A] \in B_i, t$ は Cを満たす$\}$ に入るタプル数を v_i とし，タプルの総数を N とすれば，ルール $(A \in [x_s, y_t]) \Rightarrow C$ の確信度は $(\sum_{i=s}^{t} v_i)/(\sum_{i=s}^{t} u_i)$ となり，$A \in [x_s, y_t]$ のサポートは $\sum_{i=s}^{t} u_i/N$ となる．

u_i, v_i を求めるためには，各タプル t について，$t[A]$ がどのバケットに入るか知る必要がある．このための自然な方法として，すべてのタプルをスキャンしながら，最小粒度バケットを葉とする順序付2分木を作っていく方法や，ハッシュ関数を用いる方法が考えられる．このような手法は，大規模なデータベースに対しては，(先述の年齢の例のような) バケットの数がごく少ない場合でしか実用的に動作しない．もう1つの単純な方法として，A についてリレーション全体をソートし，ソートされたデータを最小粒度のバケットに分割する方法がある．しかし実用上は，主記憶よりずっと大きなデータをソートするためには大きなコストがかかる．

以上の議論から，最も困難なケースは，最小粒度バケットの数が大きく，データベースのサイズが非常に大きい場合である．そのような例として先の銀行の預金残高の例があげられる．このような場合，我々はデータをソートすることは避けなければならないし，バケットの数も減らさなければならない．そこで次の章では，データをソートせずに数千ほどの少ない数のバケットを生成する方法について述べる．この方法は，最適ルールの十分な近似に用いることができるような，ほとんど同じ大きさのバケットを生成する．

3. バケット分割

3.1 アルゴリズム

次の方法を用いればデータ全体をソートせずに，N 個のデータをほぼ均等な M個のバケットに分割することができる[6]：

アルゴリズム 3.1

(1) S 個の標本データをデータベースからランダムに取り出す．

(2) 標本を着目する属性に関してソートする．

(3) 標本の $i(S/M)$ 番目のデータの値を p_i とし，$p_0 = -\infty, p_M = \infty$ とする．

(4) 元データの各タプル t に対して，$p_{i-1} < t[A] \leq p_i$ となる i を見つけ，t を i 番目のバケットに入れる．順序付2分木を用いれば $O(\log M)$ で i を見つけることができるので，すべてのタプルに関して $O(N \log M)$ で計算できる． □

この計算量は $O(\max(S \log S, N \log M))$ である．もし標本の大きさ S を Nと比べて十分小さくできれば，この計算量は $O(N \log M)$ となる．さらに (2) でソートを行っているため，標本が主記憶に収まる大きさになるかどうかが，実装上重要である．

3.2 標本の大きさ

標本の大きさ S を大きくすればバケットの大きさは均等になり，S を小さくすれば，ばらつきが大きくなることが予想される．そこで十分な S の大きさを知るために，次のような考察を行う．データベース中に $m = N/M$ 個のデータを含むある区間 I に注目しよう．全体からランダムに S 個の標本をとったとき，Iが含む標本点の個数 X は二項分布（二項確率 $1/M$,

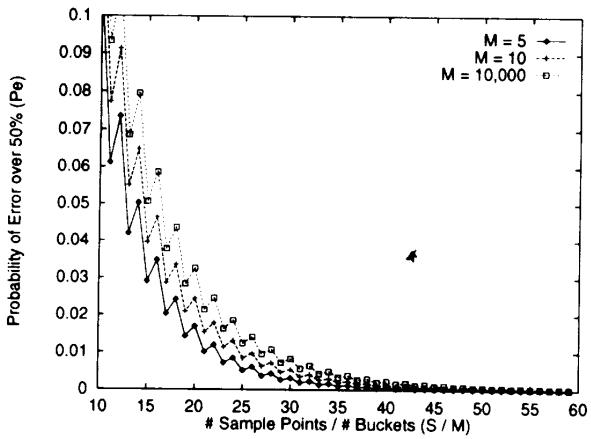

図 1 標本数と 50 % 以上の誤差を生じる確率の関係
Fig. 1 Sample size and probability of error over 50 %.

期待値 $\mu = S/M$) に従う. X が δ 以上の割合で期待値からずれる確率 $p_e = Pr(|X - S/M| \geq \delta S/M)$ を考えると p_e は M, S で決まり, データの総数 N によらない. たとえば確率的アルゴリズムの分野でよく用いられる Chernoff の定理を用いて p_e の上界を求めれば, 十分な S の大きさを決定できる. しかし, Chernoff の上界はかなり緩いため, 必要以上に大きな標本をとることになる. 一方 M, S, δ を決めれば正規分布近似を用いてかなり正確に p_e を求めることができるので, 我々は p_e の近似を元に標本数を決めることにする.

図 1 に $\delta = 0.5$, $M = \{5, 10, 10000\}$ とし, $\mu = S/M$ を変化させて p_e を求めた結果を示す (Peizer-Pratt の正規近似法を用いた). この図から分かるように, S/M を増やしていくに従って, はじめ p_e は急速に減少し, $S/M = 40$ 付近で p_e は $0.0006 \sim 0.0022$ と十分に小さくなっている. また, $S/M = 20 \sim 30$ 付近に曲線の肩があり, それ以上大きくしても p_e はあまり小さくならない. したがって, $S = 40M$ 程度とするのが良いことが分かる. このようにすると, 実際の問題では $S \ll N$ であるから, アルゴリズム 3.1 の計算量は $O(N \log M)$ となる. また通常, たかだか $M = 10^4$ で十分な精度のルールが得られるので, 標本の大きさは $4 \cdot 10^5$ 程度以下となり, 今日の計算機の主記憶に十分収まる. また 50 % 以上誤差を含むバケット数の期待値は $M = 10^4$ のとき, およそ $p_e M \approx 22$ である.

3.3 並列化の指針

アルゴリズム 3.1 で最も計算時間がかかるのは (4) ですべてのデータを走査して各タプルが属すバケットを見つける処理である. 我々が知りたいのはバケットの境界と大きさだけであるから, この処理は次のように簡単に並列化できる:

アルゴリズム 3.2

(1) あらかじめ, データベースを要素プロセッサにほぼ均等に分配しておく.
(2) 1 つのプロセッサで, アルゴリズム 3.1 の (1), (2), (3) を実行する.
(3) 各要素プロセッサで, アルゴリズム 3.1 の (4) を実行する. すなわち, データベースを走査して, 各バケットの大きさを調べる.
(4) 1 つのプロセッサで, 全プロセッサの結果を足し合わせる.

このアルゴリズムは, バケットの大きさを数える処理の間, プロセッサ間の通信をまったく必要としない. このため, プロセッサ数にほぼ比例した処理速度が期待できる.

4. 最適区間の抽出

2.3 節で述べたように, バケットが与えられれば連続するバケットをつないで, ルールに用いる区間を作ることができる. この章では, そのような区間の中から最適なものを見つけ出すアルゴリズムを説明する.

バケットの列 $B_1, B_2, \ldots, B_M$ が与えられたとき, 次のような形のルール:

$$(A \in [x_s, y_t]) \Rightarrow C$$

を生成することを考える. ここで $[x_s, y_t]$ は連続するバケット $B_s, B_{s+1}, \ldots, B_t$ を連結したものである. 任意の区間 $[x_s, y_t]$ がバケットのインデックスの対 $(s \leq t)$ で特定できる. 簡単のために, 区間 $[x_s, y_t]$ を単に $[s, t]$ と表記する. また $sup(A \in [x_s, y_t])$ は $sup(s, t)$ と, $conf(A \in [x_s, y_t] \Rightarrow C)$ は $conf(s, t)$ と表記する.

4.1 最適確信度ルール

この章での我々の目的は $sup(s, t)$ が閾値 *minsup* より小さくない対 (s, t) の中で, $conf(s, t)$ を最大とするもの (最適確信度対, *optimized confidence pair* と呼ぶ) を求めることである. もし確信度を最大とする対が複数存在するとき, サポートを最大にする対を選ぶことにする.

まず, $k = 1, \ldots, M$ に対して, 2 次元平面における点の列 $Q_k = (\sum_{i=1}^{k} u_i, \sum_{i=1}^{k} v_i)$ を考え, $Q_0 = (0, 0)$ とする. このように準備すると, 線分 $Q_m Q_n$ の傾きが $conf(m+1, n)$ を与え, Q_m と Q_n の x 座標の差が $sup(m+1, n)$ となる. したがって, 最適確信度対発見問題は x 方向に *minsup* 以上離れた 2 点で, それらが傾きが最も大きいものを発見する問題に変換される.

この問題を解くために, 我々は点を囲む凸包を用い

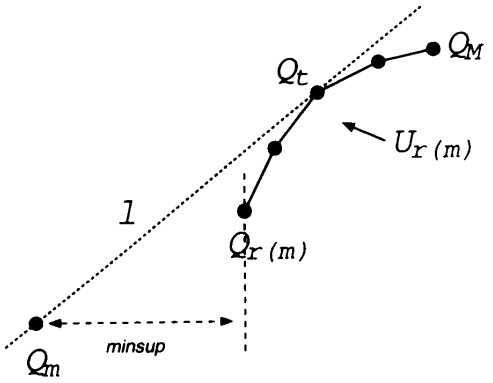

図2 Q_m と $U_{r(m)}$ の接線
Fig. 2 The tangent of Q_m and $U_{r(m)}$.

る．いくつかの用語を定義しておこう．S を点の集合としたとき，S の**凸多角形** (*convex polygon*) とは S の要素を頂点とする多角形のうち S の任意の 2 点を結んだ線分が完全にその多角形内部にあるものである．S の最も小さい凸多角形を S の**凸包** (*convex hull*) と呼ぶ．S に属する点の中で，最小の x 座標と最大の x 座標を持つ点をそれぞれ Q_{min}, Q_{max} とする．Q_{min}, Q_{max} はともに S の凸包上にある．Q_{min} から出発して凸包上を時計回りに移動して行くと Q_{max} に到達できる．このとき通る点を S の**上包** (*upper hull*) と呼ぶ．同様に反時計回りで通る点を**下包** (*lower hull*) と呼ぶ．

さて，U_m を $\{Q_m, \ldots, Q_M\}$ の上包とする．さらに $r(m)$ を

$$r(m) = \min\{i \mid sup(m+1, i) \geq minsup\}$$

とする．ここで Q_m から $U_{r(m)}$ へ引いた接線を考え，Q_t を $U_{r(m)}$ 上の接点とする (**図 2** 参照)．複数の点が接線上にある場合は，最も Q_m から遠い点を Q_t に選ぶことにする．この接線は Q_m と $U_{r(m)}$ 上の頂点を結ぶ線分のうち最も傾きの大きいものである．したがってすべての m について Q_m と $U_{r(m)}$ の接線を考え，傾きを最大とする m を選べば，最適確信度対を求めることができる．以下にこの計算を $O(M)$ で行うアルゴリズムを示す．

上包の維持

接線を求めるために，各 $m = 0, \ldots, M-1$ について $U_{r(m)}$ 上の点を時計回り，反時計回りにたどる操作が発生する．この処理を高速に行うために，上包上の隣接する点を定数時間でアクセスする方法を示す．

アルゴリズム 4.1 S, $D_0, \ldots, D_M$ を空のスタックとする．このアルゴリズムの最終的な目的は，$i = 0, \ldots, M-1$ について U_i 上の点が時計回りの順序でスタック S に積まれるようにすることである．そうすれば，U_i 上の任意の点からその隣の点に定数時間でアクセス可能である．

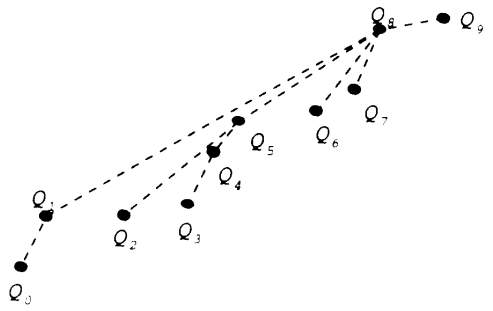

図3 上包 (例 4.1)
Fig. 3 Upper hulls (Example 4.1).

準備段階： まず，S に U_{M-1} 上の点である Q_M と，Q_{M-1} を順番に乗せる．次に，$i = M-2, \ldots, 0$ に関して，S から U_i 上の点ではなくなる点を D_i に移し，S に Q_i を乗せる．これは次の手続きにより実行される：

> 時計回りサーチ： もし Q_i と S の先頭の点が作る傾きが，Q_i と S の 2 番目の点が作る傾きより小さいか等しいならば，S の先頭の点はもはや U_i 上の点ではないので，先頭の点を D_i に移し，この検査を繰り返す．そうでなければ，Q_i と S の先頭の点の作る傾きが最大なので，Q_i を S の先頭に乗せる．

スタック D_i は各ステップで削除された点を記録するために使われる．全体としてたかだか $M-1$ 個の点が S から削除されるだけなので，時計回りサーチの時間および空間計算量はともに $O(M)$ である．この時点で S には U_0 が格納されている．

復元手順： 準備段階とは反対の順序，すなわち $i = 1, \ldots, M-1$ の順で次を繰り返す．S の先頭の点 Q_i を取り除き，D_i の内容を順に S に移す．これにより S に U_{i+1} の点が格納される． □

例 4.1 **図 3** のような点 $Q_0, \ldots, Q_9$ について考える．Q_i から Q_9 にかけての点線は $\{Q_i, \ldots, Q_9\}$ の点が作る上包である．$i = 1, \ldots, 9$ について，アルゴリズム 4.1 は S, D_i を次のように変化させながら動作する：

$i =$	8	7	6	5	4	3	2	1	0
	Q_8	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2		Q_0
	Q_9	Q_8	Q_8	Q_8	Q_5	Q_4	Q_5	Q_1	Q_1
		Q_9	Q_9	Q_9	Q_8	Q_5	Q_3	Q_8	Q_8
					Q_9	Q_9	Q_9	Q_9	Q_9

		Q_7	Q_6			Q_4	Q_5	
						Q_3	Q_2	
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	

□

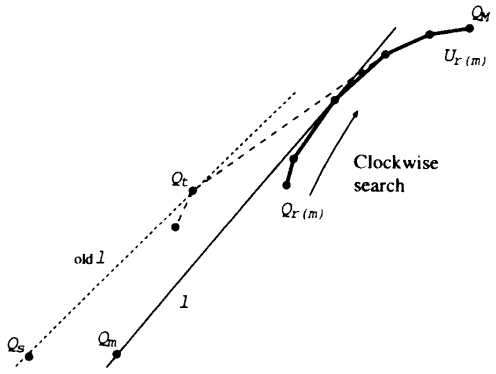

図 4 時計回りサーチ
Fig. 4 Clockwise search.

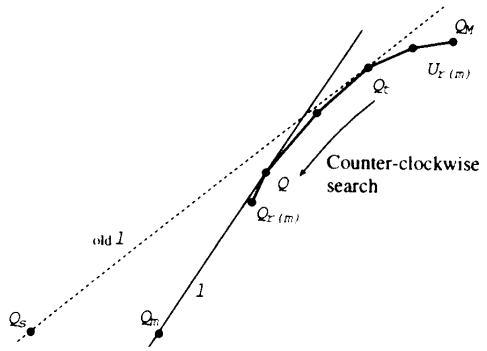

図 5 反時計回りサーチ
Fig. 5 Counter-clockwise search.

最大傾き接線の計算

以下のアルゴリズムでは $m = 0, \ldots, M-1$ について Q_m と $U_{r(m)}$ が作る接線を考え，その中で最大の傾きを持つものを計算する．

アルゴリズム 4.2 まず，与えられた $Q_0, \ldots, Q_M$ に対して，アルゴリズム 4.1 を $O(M)$ 時間かけて実行し，$m = 0, \ldots, M-1$ について $U_{r(m)}$ を得る．以下で l は直前に求めた接線を保持する変数で，不要な接線を求めないために用いる．

$m = 0$ のとき： $U_{r(0)}$ を時計回りサーチで（スタックの上から下に調べて），Q_0 との接線を見つけ，それを l とする．

$m = 1, \ldots, M-1$ のとき： もし Q_m が l より下にないとき，Q_m と $U_{r(m)}$ の接線は必ず l の傾きより小さくなり，最大の傾きを持つことはないので，l を更新しない．もし Q_m が l より下にあるとき，$l = Q_s Q_t$ $(s < m)$ として，次のどちらかを実行する：

- l が $U_{r(m)}$ と接しないとき（すなわち $t < r(m)$）：Q_t は $U_{r(m)}$ 上にない．時計回りサーチで Q_m と $U_{r(m)}$ の接線を見つけ l とする（**図 4** 参照）．
- l が $U_{r(m)}$ と接するとき（すなわち $t \geq r(m)$）：Q_t は $U_{r(m)}$ 上にある．次の反時計回りサーチにより Q_m と $U_{r(m)}$ の接線を見つけ l とする：
  反時計回りサーチ： Q_t から，$U_{r(m)}$ 上の点 Q を反時計回りの順に調べて，もし $Q_m Q$ の傾きが Q_m と次の点との傾きより小さければ，次の点に移って繰り返す．そうでなければ，$Q_m Q$ が Q_m と $U_{r(m)}$ の接線である（**図 5** 参照）．

最後に，求めた接線の中で最大の傾きを持つものを出力して終了する． □

補助定理 4.1 アルゴリズム 4.2 は，$O(M)$ 時間で終了する．

証明 時計回りサーチと反時計回りサーチは各々上包上の各辺をたかだか 1 回しか通過しないことが示せる．辺の総数は M 以下なので計算量は $O(M)$ である．

まず，時計回りサーチについて考えよう．$m = 0$ のサーチで通る辺はすべて初めての辺である．$m = 1, \ldots, M-1$ のステップで l と $U_{r(m)}$ が接しないとき調べられるのは，$Q_{r(m)}$ から次のような条件を満たす点 Q_{next} を越えないことが凸包の性質から分かる：

$$next = \min\{i \mid i \geq r(m), Q_i \in U_{r(s)}\}.$$

Q_{next} の手前までの辺は，前回時計回りサーチが起こった可能性のある $U_{r(s)}$ 上には存在しなかった辺であるから，当然時計回りサーチで初めて通る辺である．

次に反時計回りサーチについて考える．反時計回りサーチが操作する辺はすべて，最後に実行された時計回りサーチで新たに出現した辺である．したがって反時計回りサーチも上包上の各辺をたかだか 1 回しか通過しない． □

以上の議論より次の定理が得られる．

定理 4.1 すべての最適確信度対は $O(M)$ 時間で計算可能である．

4.2 最適サポートルール

この章での我々の目的は，$conf(s,t)$ が閾値 $minconf$ より小さくない対の中で $sup(s,t)$ を最大とする対 (s,t) を求めることである．このような対のことを**最適サポート対**（*optimized support pair*）と呼ぶ．

すべての $j < s$ に対して $conf(j, s-1) < minconf$ のとき，s を**有効**と呼ぶことにすると，次の補助定理が成り立つ．

補助定理 4.2 もし (s,t) が最適サポート対ならば，s は有効である．

証明 s が有効でないとすると，$conf(j, s-1) \geq minconf$ とする j が存在する．(s,t) は最適サポート対だから $conf(s,t) \geq minconf$ である．よって，$conf(j,t) \geq minconf$．$sup(j,t) > sup(s,t)$ だから，これは (s,t) が最適サポート対である条件（$sup(s,t)$ が最大）と矛盾する． □

この補助定理から，有効であるインデックスを探し，その中から最適なものを見つければよいことが分かる．$w = \max_{j<s} \sum_{i=j}^{s-1}(v_i - u_i \times minconf)$ とすると，$w < 0$ のときに限り s は有効である．次のアルゴリズムはすべてのインデックスに対する w を $O(M)$ で求めることができる．

アルゴリズム 4.3

1 は有効である．
$w := 0$
for $s := 2$ to M begin
  if ($w < 0$ and $v_{s-1} - u_{s-1} \times minconf \geq 0$)
    then $w := v_{s-1} - u_{s-1} \times minconf$
    else $w := w + v_{s-1} - u_{s-1} \times minconf$
  if ($w < 0$) then s は有効である．
end. □

$conf(s,t) \geq minconf$ とする最大のインデックス t を $top(s)$ と表すとすると，後は $\sum_{i=s}^{top(s)} u_i$ を最大とする s を求めればよい．

補助定理 4.3 s, s' がともに有効で $s < s'$ ならば，$top(s) \leq top(s')$ である．

証明 s' が有効であるから，$conf(s, s'-1) < minconf$．$top(s)$ の定義から，$conf(s, top(s)) \geq minconf$ である．したがって，$conf(s', top(s)) \geq minconf$．このことから，$top(s) \leq top(s')$． □

この性質から，有効なインデックスのリスト $(s_1, \ldots, s_q)$ と全インデックスのリスト $(1, \ldots, M)$ を用意しておけば，これらのリストを（行ったり来たりせずに）逆方向に交互に走査して，$top(s_i)$ を見つけることができる．そのアルゴリズムを次に示す：

アルゴリズム 4.4

$i := M$
for each j from q to 1
  if ($conf(s_j, i) < minconf$) then $i := i - 1$
    else begin
      $top(s_j) := i$
      $j := j - 1$
    end. □

このアルゴリズムでは，あらかじめ $F(j) = \sum_{i=j}^{s} v_i - minconf \sum_{i=j}^{s} u_i$ の値を計算しておいて，$F(i) < F(s_j - 1)$ と $conf(s_j, i) < 0$ が同値であることを利用する．このようにすれば，$conf(s_j, i) < minconf$ の判定が定数時間でできる．アルゴリズム 4.3, 4.4 はともに $O(M)$ 時間で終了するので，次の定理が得られる．

定理 4.2 すべての最適サポート対は $O(M)$ 時間で計算できる．

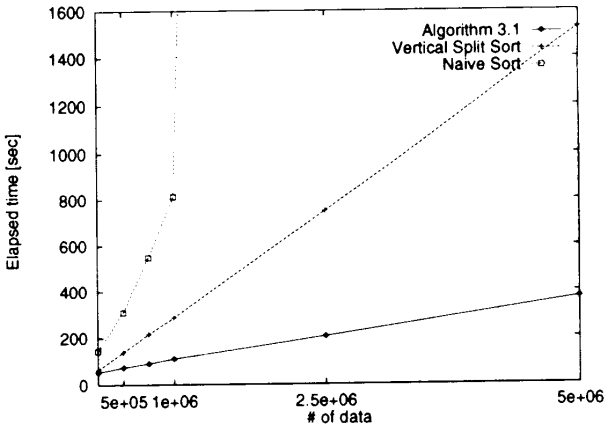

図 6 バケット分割の性能
Fig. 6 Performance for making buckets.

5. 性能評価

我々は，提案したアルゴリズムを C++ 言語で実装し，IBM Power Series 850（CPU: PowerPC 133 MHz，主記憶：64MB，DISK: 3.5" 1.2 GB IDE，OS: AIX 4.1.3）の上で性能を実測した．

5.1 バケット分割

我々のアルゴリズム 3.1 と比較する方法として，データ全体を素直にソートする方法の他に，データを数値属性ごとに垂直分割し，属性ごとにソートしてバケットの境界を求める方法が考えられる．このように分割すれば今日の計算機では数千万件程度までのデータならば主記憶に収めることができる．

そこでこれらの方法を 8 個の数値属性と 8 個の 2 値属性をタプルとする（1 タプルあたり 72 バイト）ランダムに生成されたテストデータを用いて比較した．バケット分割は，8 個の数値属性すべてに関して 1,000 個のバケットに分割し，すべての 2 値属性に関して各バケットに含まれる 2 値属性値が 1 となるデータ数を数えることとする．垂直分割する方法に関してはあらかじめデータは分割されているものとする．

図 6 にデータ数を $5 \cdot 10^5$ から $5 \cdot 10^6$ まで変化させたときの実行時間を示す．全体をソートする方法は，百万件以上ではソートの際にデータが主記憶に収まらずまったく実用にならない．一方，我々の方法の実行時間は，データ量に対してほぼ比例しており，データを垂直分割する方法と比べても高速で，データが多くなるにつれてその差は大きくなる．

5.2 最適ルールの発見

最適確信度ルールを発見するアルゴリズム 4.1, 4.2 と，最適サポートルールを発見するアルゴリズム 4.3, 4.4 の性能を評価するために，これらの方法と，素直

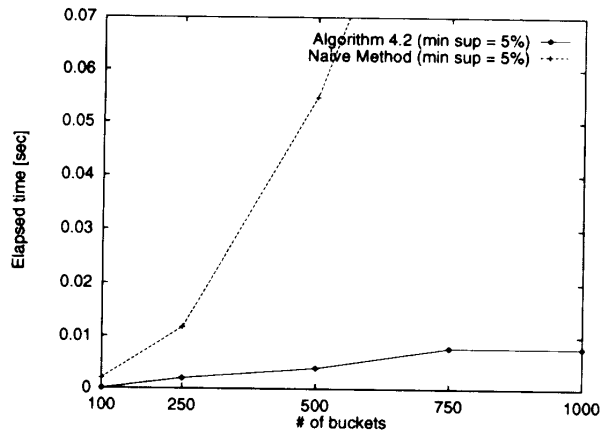

図7 最適確信度ルール発見の性能
Fig. 7 Performance for finding optimized confidence rule.

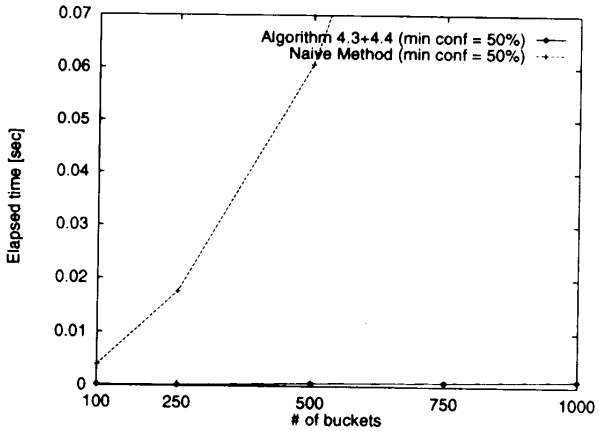

図8 最適サポートルール発見の性能
Fig. 8 Performance for finding optimized support rule.

にすべての区間のサポートと確信度をしらみつぶしに調べて最適なものを選ぶ方法とを比較した.

図7と**図8**に,バケット数を変えながら,最適確信度ルールを $minsup = 5\%$ として発見するのにかかる時間,および最適サポートルールを $minconf = 50\%$ として発見するのにかかる時間を各々測定した結果を示す.どちらの最適ルールも,素直な方法に比べて我々の方法がバケット数がごく少ないときから何倍も速く,バケット数を多くするに従いその差はますます広がっている.我々の方法の実行時間は,バケット数をさらに増やしてもバケット数にほとんど比例しており,バケット数が 10,000 のとき,最適確信度ルールおよび最適サポートルールを各々 0.1 秒, 0.01 秒程度で計算する.一方,素直な方法ではともに 20 秒以上かかる.

6. まとめ

我々はこの論文で,1つの数値属性を条件部に持つ結合ルールを考え,最適なサポートや確信度を持つルールを発見する問題を提案し,この問題を線形時間で解くアルゴリズムを説明した.さらに提案したアルゴリズムを実装して実験した結果,理論的計算量が現実にも達成できたことを示した.

実際のデータマイニングの作業では,ユーザが興味がある数値属性と2値条件のすべての組合せに対して最適ルールを求めることになる.また,いったんバケット分割したデータに対して,閾値 $(minconf, minsup)$ を変えながら繰り返し最適ルールを求めて,その中から最も"面白い"ルールを探すといった対話的な使い方を考えると,我々のアルゴリズムで達成された高速性は重要である.

今後の課題として,本論文では確信度とサポートの最適化を扱ったが,それ以外の目的評価関数を最適化する問題を考えることができる.利益の最大化といった,ユーザの興味に直結した最適化ができれば,データマイニングとして非常に有効である.

また,本論文では条件部の数値属性を1つと限ったが,データの特徴をとらえるためには,1つの数値属性だけで十分とはいえない.そこで,条件部を複数の数値属性が作る領域に拡張(多次元化)する問題を解くことは重要な課題である.我々は文献 7) で条件部の数値属性が2つのときの問題について論じている.

参考文献

1) Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B. and Swami, A.: An Interval Classifier for Database Mining Applications, *Proc. 18th VLDB Conference*, pp.560–573 (1992).
2) Agrawal, R., Imielinski, T. and Swami, A.: Database Mining: A Performance Perspective, *IEEE Trans. Knowledge and Data Engineering*, Vol.5, No.6, pp.914–925 (1993).
3) Agrawal, R., Imielinski, T. and Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, *Proc. ACM SIGMOD Conference on Management of Data*, pp.207–216 (1993).
4) Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules, *Proc. 20th VLDB Conference*, pp.487–499 (1994).
5) Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J.: *Classification and Regression Trees*, Wadsworth (1984).
6) Floyd, R. and Rivest, R.: Expected Time Bounds for Selection, *Comm. ACM*, Vol.18, pp.165–172 (1975).
7) Fukuda, T., Morimoto, Y., Morishita, S. and Tokuyama, T.: Data Mining Using Two-

Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, to appear in *Proc. ACM SIGMOD Conference on Management of Data* (1996).
8) Fukuda, T., Morimoto, Y., Morishita, S. and Tokuyama, T.: Mining Optimized Association Rules for Numeric Attributes, to appear in *Proc. ACM Symposium on Principles of Database Systems* (1996).
9) Han, J., Cai, Y. and Cercone, N.: Knowledge Discovery in Databases: An Attribute-Oriented Approach, *Proc. 18th VLDB Conference*, pp.547–559 (1992).
10) Ng, R.T. and Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining, *Proc. 20th VLDB Conference*, pp.144–155 (1994).
11) Park, J.S., Chen, M.-S. and Yu, P.S.: An Effective Hash-Based Algorithm for Mining Association Rules, *Proc. ACM SIGMOD Conference on Management of Data*, pp.175–186 (1995).
12) Piatetsky-Shapiro, G.: Discovery, Analysis, and Presentation of Strong Rules, *Knowledge Discovery in Databases*, pp.229–248 (1991).
13) Piatetsky-Shapiro, G. and Frawley, W.J. (Eds.): *Knowledge Discovery in Databases*, AAAI Press (1991).
14) Quinlan, J.R.: Induction of Decision Trees, *Machine Learning*, Vol.1, pp.81–106 (1986).
15) Quinlan, J.R.: *C4.5: Programs for Machine Learning*, Morgan Kaufmann (1993).
16) Stonebraker, M., Agrawal, R., Dayal, U., Neuhold, E.J. and Reuter, A.: DBMS Research at a Crossroads: The Vienna Update, *Proc. 19th VLDB Conference*, pp.688–692 (1993).

(平成8年1月31日受付)
(平成8年4月12日採録)

福田 剛志（正会員）

1966年生．1991年早稲田大学大学院理工学研究科修士課程修了．同年日本アイ・ビー・エム（株）入社．現在，東京基礎研究所副主任研究員．ソフトウェアのリバースエンジニアリング，オブジェクト指向データベース，データマイニングなどの研究に従事．ACM会員．