修　士　論　文

ブースティング手法を用いた
分類システムの精度向上

指導教官　　Ho Tu Bao　教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

150003　綾川 聡司

審査委員：　Ho Tu Bao　教授（主査）
石崎　雅人　助教授
佐藤　賢二　助教授
林　　幸雄　助教授

2003 年 2 月

目　次

図　目　次

表　目　次

第 1 章
はじめに

1.1 研究の背景と目的

近年、情報技術の発展にともないデータの量は持続的に増大している。そのため情報産業では、巨大なデータの中から有用な情報、知識を取り出すこと（データマインニング）が必要になってきた。データマインニングとは、様々なタイプの巨大なデータの中から、科学調査、企業経営、生産管理および市場分析に活用する事ができる知識を抽出する方法である。データマインニングは、データの中から知識を取り出す方法であり KDD（Knowledge Discovery in Databases）とも呼ばれている。

データマインニング手法には、様々な手法があるが本研究では、過去のデータを分類することにより未知のデータを予測する分類システムについて注目した。良い分類は未知の事例に対する適切な予想を与える事になる。したがって、分類システムの精度を向上させる研究が行われている。その研究の一つのアプローチとして、分類システムがデータから抽出した分類規則（仮説）を組み合わせることにより、精度の高い分類規則を作成する手法がある。その手法はアンサンブル学習と呼ばれ、ブースティングやバッキングといった手法がある。

本研究ではブースティング手法を使った分類システムの精度向上について注目した。また、ブースティング手法は様々な種類の手法があるが、分類システムに適用可能な AdaBoost アルゴリズムを用いることにした。このアルゴリズムは、様々な学習システムに適用した実験で良い結果を得ている。しかし、いくつかの問題もある。たとえば、「適用するシステムの精度が低すぎる」、「適用するルールが複雑すぎる」、「学習するデータに間違ったデータが多く存在する」などの場合はあまり効果的では

ないことが分っている。現在のブースティング研究の一つに、これらの問題を解決するために学習システムとブースティング手法との関係に関する研究がなされている。本論文ではこの研究に注目している。

1.2 本研究の目的

本研究の大きな目的は、分類システムのエラー率の減少させることである。そのため分類システムの精度を向上させる方法の一つであるブースティング手法の調査、ブースティング手法と分類システムとの関係について調査、ブースティングの改良の提案を目標として研究を進めた。その方法として、決定木を用いた分類システムである C4.5 が生成する精度の異なる 2 種類の決定木（ひとつは“un-pruned tree”と呼ばれる規則、も一つはそれを一般的なデータに対応出来るようににした規則“pruned tree”）に対して AdaBoost を適用し、その精度を比較することを行った。また、AdaBoost を改良したアルゴリズムを 2 種類考案した。改良を行ったシステムも C4.5 の 2 種類の決定木に適用しそれぞれ比較を行った。

1.3 本論文の構成

本論文は、7 つの章よりなる。 2 章は分類システム、特に決定木を用いた分類システムである C4.5 について述べた。また、分類システムの評価方法についても紹介した。3 章はブースティングについての紹介をした。また、ブースティング手法の一つである AdaBoost のアルゴリズムについて述べた。4 章は AdaBoost の改良についての提案について述べる。また、AdaBoost を改良したアルゴリズムを 2 種類紹介した。5 章は本研究で開発したシステムの概要について記した。6 章では実験で使用したデータ、具体的な実験の進め方、実験結果、結果に対する考察を記した。7 章では本研究に対する結論を述べた。

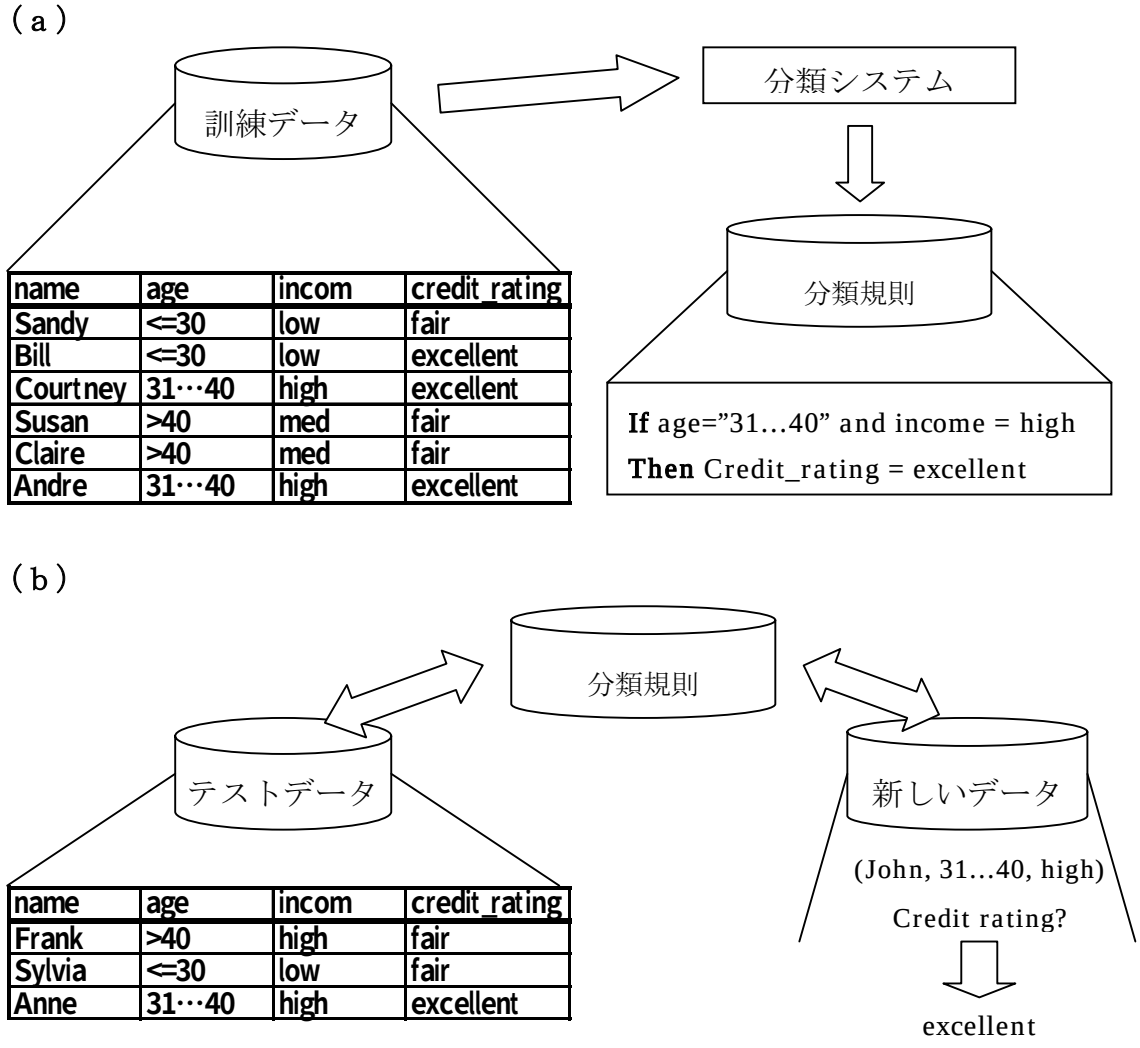
第 2 章
分類システム

2.1 分類システムとは

分類システムは学習用データセット(訓練データ)から分類規則を構築し、それを用いて未知データセットに予測結果(クラスラベル)を与えるシステムである。分類システムには、決定木、ニューラルネットワーク、Nearest Neighbor などがある。

データの分類には、2 つのステップがある。まず、データのラベルまたはコンセプトに基づいた分類規則を構築する。その分類規則は、属性によってわけられているデータベース内を分析することによって得られる。また、データベース内の各データには、あらかじめ設定された一つの属性（クラス属性)、ラベルが設定されている。このようなデータ集合をデータセット、事例、オブジェクトなどと言う。このような各訓練データにクラスラベルがすでに与えられている物を使用して学習を行うシステムは、教師付き学習と呼ばれている。一方、このような学習とは別に教師無し学習（クラスタリング）と呼ばれる分類システムがある。これは、データベースの各データに決められたクラスラベルは決められていない。また、分類されるべきクラスの属性や数などは、学習課程で決められていく。本論文では、教師付き学習についてのみ取り上げている。

通常、学習されたモデルは分類ルールの形で示されている。たとえば、決定木または数式などである。ここで、**図 2. 1（a）**をもとに例を挙げる。まず学習用のデータセットは、顧客のクレジット情報である。この場合クラス属性は“credit_rating”に設定されているので、作成されるルールは“fair”または“excellent”を導き出している。作成されたルールは、未知のデータの予測に使用される。



name	age	incom	credit_rating
Sandy	<=30	low	fair
Bill	<=30	low	excellent
Courtney	31…40	high	excellent
Susan	>40	med	fair
Claire	>40	med	fair
Andre	31…40	high	excellent

name	age	incom	credit_rating
Frank	>40	high	fair
Sylvia	<=30	low	fair
Anne	31…40	high	excellent

(a) 学習：分類システムは、訓練データを分析し分類規則を生成する。

(b) 分類：テストデータは、分類規則の評価に用いられる。もしその評価が満足の行くものっだったならば、新しいデータを適用した場合も満足の行く予測が出来る。

図２．１　データ分類のプロセス

つぎのステップは、**図 2．1**（b）である。作成された分類規則は、データの分類に使用される。**図**の左側の作業は、分類規則の予測精度が評価されている。分類システムの評価については、**2.3** で説明する。もし、訓練データを用いて評価をした場合は、その分類規則はそのデータに特化した規則なのでこれは、評価とはいえない。よって、ここで用いるテストデータは、訓練データとは異なった物を使用する。もし、

分類器の評価が満足のいく物であったならば、未知のデータ（クラスラベルが分らないデータ）のクラスラベルを高い精度で予測することが出来る。

分類の方法は、様々な方法があるが、本論文で使用する決定木による分類システム（**C4.5**）について紹介する。

2.2 決定木による分類（C4.5）

決定木とは、**図 2.2** のような木構造のフローチャートである。枝の分かれ目（node）がデータの属性、そこの枝（branch）がその属性の値、そして葉の部分（leaf）がクラスのラベルとなっているとくに、一番元になるノードが根の部分（root）となっている。**図 2.2** は、コンピュータをほしいと思っている顧客がコンピュータを買うか買わないかと言うコンセプトで作成された決定木である。この決定木を使用して未知のデータを予測するときは、まずルートの部分（ここでは “age”）から分類を始め次のノードへと分類を進め、最後のリーフの部分で結果がでる。このように決定木は、ニューラルネットワークなどとは異なり、非常にわかりやすい“ルール“を作成することが出来る。作成されたルールは、「IF…THEN…」のような一般的な言語で簡単に表すことが出来る。

分類器を使用した分類システムには、ID3、CART、C4.5 などがある。本論文では、**C4.5** を分類システムとして使用する。そこでそれの動作について、2.2.1 で決定木の作成を説明する。また、この分類システムは、木の枝刈りを行う。それは、作成した木は枝の数が多く訓練データに非常に特化したルールになっている(過学習)からである。この方法を 2.2.2 で説明をする。

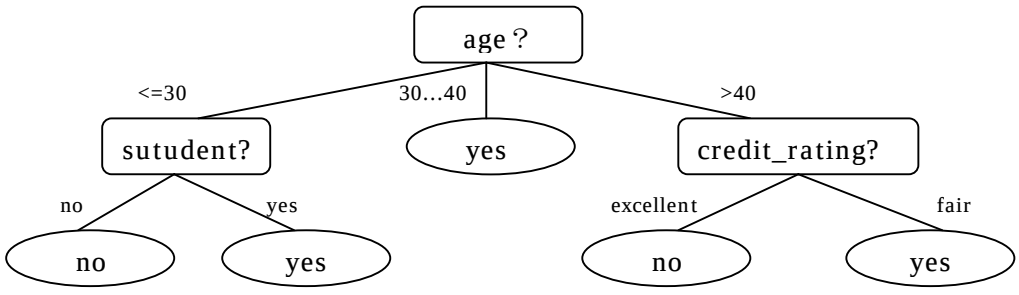


図２．２　決定木

2.2.1　決定木の作成

決定木作成は、まずノードの決定から始まる。もし探査しているデータセット内のクラスラベルがすべて同じ種類だったならば、そのノードはリーフになる。そうでない場合は、そのノードに当てはまる“属性”を選択する。この作業を繰り返し行ってすべてリーフにたどり着いたら学習を終了させる。

C4.5 は、属性の決定のために利得比(Gain Ratio)という評価関数を用いて選択をする。この方法について以下で説明をする。

集合 s で構成された集合 S について考える。集合 S のクラス属性は、m 個のクラス（$C=\{C_1,...,C_m\}$）を持っているとし、s_iは集合内のクラスC_iをサポートするデータの数とする。そうすると、集合 S を表すために必要な情報量は、

$$I(s_1,s_2,...,s_m)=-\sum_{i=1}^{m} p_i \log_2(p_i) \quad \cdots\cdots\cdots \quad (2.1)$$

となる。ここで、p_iは集合 S の中からC_iのデータセットを選び出す確率である（s_i/S）。また、log の底が 2 となっているのはビットにエンコードされているからである。（$-\log_2(8)$ならば 3 ビット）

つぎに n 個の値を持つ属性 A（$A=\{a_1,...,a_n\}$）について考える。属性 A は、集合 S を n 個のサブセット$\{S_1,...,S_n\}$に分けることが出来る。ここでの集合S_jは、集合 S 中の属性 A のa_jを持つ物である。ここで属性 A がノードの属性となった場合は、これらのサブセットは集合 S ノードから枝分かれして別のノードになる。次の作業は、属性 A がノードの属性となった場合のエントロピーを求める。ここで、s_{ij}をサブセットS_j内のC_iをサポートするデータの数とするとエントロピーは

$$E(A)=\sum_{j=1}^{n} \frac{s_{1j}+...+s_{mj}}{s} I(s_{1j}+...+s_{mj}) \quad \cdots\cdots \quad (2.2)$$

ここでの$(s_{1j}+...+s_{mj})/s$は、集合 S 内でのサブセットS_jの頻度に対する重みづけである。また、$I(s_{1j}+...+s_{mj})$は、サブセットS_jを表すために必要な情報量であり、式（2.1）と同様の方法で求めることが出来る。さらに、上記の 2 式を使用して属性 A をノードとした場合の利得(Gain)を求める。これは、次のようになる。

$$Gain(A)=I(s_1,s_2,...,s_m)-E(A) \quad \cdots\cdots\cdots \quad (2.3)$$

C4.5 の前進である ID3 という分類システムでは、これを評価関数としてもちいて分

類を行っていた。これを評価関数として利用することで良い結果が得られるが、多数の値をとる属性を重視する傾向がある。たとえば、リストの中の名前などである。これをノードとしてしまうと１事例からなるたくさんの部分集合が出来てしまう。これを回避するために C4.5 では一種の正規化を行う。これは、多数の値を取ることによって得られた利得部分を調整することである。ある事例に関して、それがどのクラスに属するかではなく、そのテスト結果自体を伝えるメッセージの情報量を考える。これは、以下の式で表すことが出来る。

$$Split\_info(A) = -\sum_{i=1}^{n} \frac{S_i}{S} \log_2 \frac{S_i}{S} \quad \cdots\cdots\cdots \quad (2.4)$$

これは、集合Ｓをｎ個の部分集合へ分割することによって得られる全情報量を表している。一方情報量利得(Gain)は、クラス分けに関わる部分の情報量をあらわす。よって、属性Ａがノードの属性となった場合の利得比（Gain Ratio）は式２．３と式２．４を用いて

$$Gain\_Ratio(A) = Gain(A) \Big/ Split\_info(A) \quad \cdots \quad (2.5)$$

となる。

C4.5 は、このような順序で集合内のすべての属性について利得比を計算しもっとも高い評価がでた属性をノードの属性として選んでゆく。

2.2.2　木の枝刈り（Tree Pruning）

作成した決定木は、訓練集合を分類するために非常に多く枝分かれし多岐になっている。これは、この集合にのみ特化(過学習)した木である。未知のデータにも対応できるように複雑になりすぎた木の枝を刈る作業が必要になる。

では、どのように枝を刈るのだろうか。決定木の枝刈りの方法は、２種類の方法がある。

１つ目の方法は、“pre-pruning”と呼ばれ、それ以上訓練事例の集合を分割しないことを決定する方法である。これは、単純化すれば不要になる構造を作るための時間を浪費しないという特徴がある。典型的な方法では、部分集合を分割するもっとも良い方法を調べ、統計的な重要性、情報利得、誤り減少性、等々を評価する。もし、こ

の評価が何らかの閾値より低くなれば、その分割は却下される。そして、その部分集合に対する木は最適な葉と言うことになる。しかし、この方法は閾値の設定がとても難しい。決定木を作成するとき、閾が高すぎれば単純すぎる気になってしまい、低すぎれば全くこの方法が反映されないと言った問題がある。

２つ目の方法は、“post-pruning”とよばれ、構築された構造のいくつかを過去にさかのぼって削除する方法である。この方法は、通常どおり木を作成してから、作成された木の順応しすぎた部分を刈り取っていく。あとに切り捨てられる木の部分を作るために費やされる計算量は本質的な問題となる。しかし、木を生長させたあとで枝刈りを行うことは、時間はかかるがより信頼できる方法である。

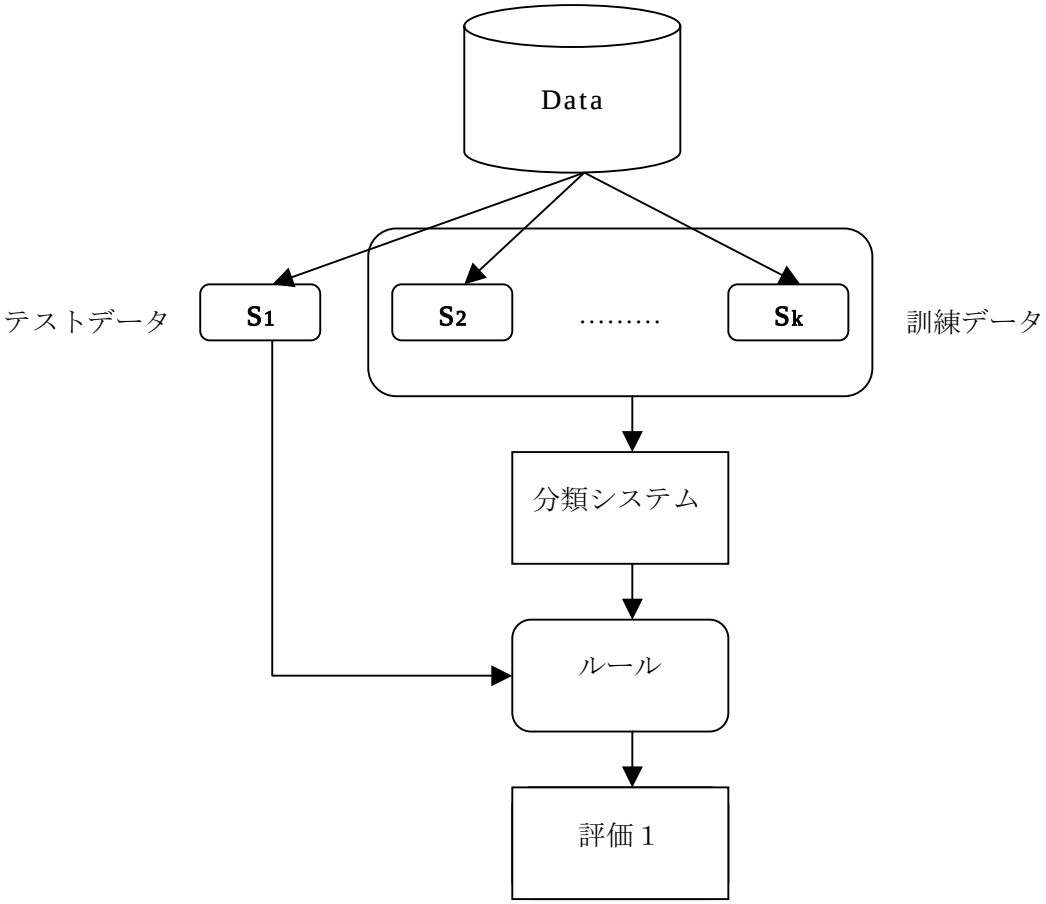
2.3　分類システムの評価方法

分類システムの正確性を見積もることは、その分類システムが作成したルールが未知のデータにたいする予測精度を知るためにも重要である。例をあげると、以前のセールスデータを用いて分類システムに訓練させ顧客の買う物を予測するとき、その予測が実際の顧客にたいしてどの程度の信頼度があるかを調べる時や、分類システムの性能を比較するときなどは、分類システムの評価はとても重要になる。ここでは、分類システムの一般的な評価方法、**holdout** と **cross-validation** を紹介する。

Holdout は、まず与えられたデータをランダムに訓練データとテストデータの２つに分割する。特に、訓練データは前データの３分の２を割り当て、残りをテストデータに割り当てる。そして、訓練データを分類システムに学習させテストデータと共に作成されたルールの評価を行う。訓練に用いたデータを使用して評価をするよりは、良い評価ができる。さらに、k回同じ作業を繰り返し行いその平均値をその分類システムの評価とする。

k-fold Cross-validation は、与えられたデータをランダムにｋ個の同量のデータセットに分割する。分割されたデータセットを$S_1, S_2, \ldots, S_k$とする。さらに次のような法則で、訓練とテストを k回繰り返し行う。i回目の作業でデータセットS_iをテストデータとして用いる。そして、残りを訓練データとする。訓練データを用いて分類システム学習をさせる。作成されたルールは、テストデータを用いて評価される。ここま

での作業を繰り返し行い。それらの結果の平均が分類システムの評価となる。また、通常の分類システムの評価では、10-fold cross-validation が進められている。本論文では、この評価法を 10 回行うことにより分類システムを評価した。



k-fold cross-validation の 1 回目に行われる作業を図に示した。この作業を k 回行い評価の平均が分類器の評価となる。

図 2．3：k-fold cross-validation

第 3 章
ブースティング(Boosting)

3.1 ブースティングとは

ブースティングに基本的な考え方は、ルールを組み合わせてより良いルールを作るという物である。ルールを $\{h_1, h_2, \ldots, h_T\}$ とすると組み合わされたルールは次のように表すことが出来る。

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

α_t は h_t の係数となっている。また、α_t, h_t はブースティングの学習課程で計算される。このようなアルゴリズムはアンサンブル学習アルゴリズムと呼ばれている。ほかに、**Bagging** や **Arcing** などのアルゴリズムがあげられる。

ブースティングのルーツはＰＡＣ学習研究にある。その研究の中で、Kearns と Valiant が、「ランダムより高い精度の学習機械を組み合わせることによってそれよりも良い学習機械を作ることが可能である」と言うことを立証した事から始まった。1989 年，Schapire によって理論的に保証されたブースティングが考案された。この初期のブースティングは、ニューラルネットワークをベースとした OCR に適用した実験が行われている。しかし、現実にある様々な学習アルゴリズムへの適用には様々な問題を抱えていた。

1995 年，Freund と Schapire によって現実にある学習アルゴリズムへの適用が可能な *AdaBoost* (図 2.1)が開発された。このアルゴリズムの動作は以下の節で説明する。このアルゴリズムは、理論上では学習を進めていくうえで過学習をさけることが出来るとされているが、訓練データにノイズが入っていると余りよい結果が得られない事

が実験によって示されている。

最近のブースティング研究は、より効果的なアルゴリズムの開発、ベースとなる学習機械の性能との関係の解明などがある。本研究では、ベースとなる学習機械の性能と AdaBoost との関係の分析に注目した。また、AdaBoost を改良しそれとの関係についても分析を行った。以下に、AdaBoost アルゴリズム、改良について述べる。

3.2 AdaBoost

AdaBoost（**Ad**aptive **Boost**ing）は、初期のブースティングの問題点を解決すべく Freund と Schapire によって提案されたアルゴリズムである。**AdaBoost** は、**図３．１**に示すようなアルゴリズムであり、その基本動作は以下で説明する。まず、入力として訓練データ $\{(x_1,y_1),\ldots,(x_m,y_m)\}$を受け取る。ここで、各 x_iは一定の事例空間 $\overline{X}$ に属しており、また各 y_iは一定のラベル集合 $\overline{Y}$ に属している。そして、ベースとなる学習機械（BaseLearner）を呼び出すラウンドを T回繰り返す（$t=1,..T$）。ここでポイントとなるアイディアは、訓練データ上に定義された確率分布（または重み）によるリサンプリングを用いると言うことである。ラウンド tにおけるこの分布による事例 i 上の重みを $D_t(i)$ と書く。これらの重みの初期値はすべて等しく設定するが、各ラウンドにおいて謝って予測された事例の重みが増やされ、BaseLearner がより難しい事例に集中して学習するようになっていく。

AdaBoost のなかでの BaseLearner の働きは、確率分布 D_tに対して適した弱仮説 $h_t: X \rightarrow Y$.を見つけ、出力することにある。ここで弱仮説 h_tの重要度は、D_tによる誤り確率

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$$

により測られる。ここでの D_tは、BaseLearner に学習させる際使用した事例の分布であることに注意する。

The algorithm **AdaBoost**

Input: m 個の訓練データ $\{(x_1,y_1),\ldots,(x_m,y_m)\}$ ラベル集合は、 $y_i \in Y=\{1,\ldots,k\}$ とする。

ベースとなる学習機械： **BaseLearner**

トライアル回数 T の設定

Step1: $D_1(i)=1/m$ によって各事例に対して重みを初期化

Step2: $t=1,2,\ldots,T$ に対して

1. 確率分布 D_t を用いて BaseLearner による学習
2. 仮説 h_t: $X \to Y$.を得る。
3. D_t による誤り確率 h_t: $\varepsilon_t = \sum_{i:h_t(x)\neq y_i} D_t(i)$.を計算する。

   もし $\varepsilon_t > 1/2$ ならば、$T=t-1$ として最初のステップに戻る。
4. 仮説の重要度 $\beta_t = \varepsilon_t/(1-\varepsilon_t)$.を計算。
5. 重みの更新 D_t: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i)=y_i \\ 1 & \text{otherwise} \end{cases}$

   Z_t は、正規化定数。

Step3: 最終仮説

$$h_{fin}(x) = \arg\max_{y\in Y} \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}.$$

図 3. 1 : AdaBoost アルゴリズム

弱仮説 h_t が得られるたびに図 2.1 に示されるように β_t を設定する。これは、弱仮説 h_t の重要度を示している。ここで、もし $\varepsilon_t \leq 1/2$ であれば $\beta_t \leq 1$ となり、一般性を失わずに仮定することが出来る。また、ε_t が小さければ小さいほど、β_t は小さくなる。次に、確率分布 D_t の更新が**図 3．1** に示すような規則で行われる。この更新規則により、弱仮説によって正しく分類された事例に対し重みを減らし、間違って分類された事例に対し重みを減らしていく。このように、重みは難しい事例に対して集中していく。

最終仮説 h_{fin} は、こうして得られた T 個の弱仮説 h_t とその重要度 β_t を用いて、重み付き多数決として得られる。

3.3 AdaBoostの特徴

このアルゴリズムは、現実に適用するにあたって多くの優れた性質を持っている。たとえば、単純なアルゴリズムであり、プログラムが簡単であり、計算も効率的である。トライアル回数Tをのぞけば調整の必要なパラメータもない。また、**BaseLearner**にたいする詳しい知識を必要とせず、仮説を発見するアルゴリズムであれば任意の物と組み合わせることが出来る。しかも、**BaseLearner**の予測精度と訓練データのサイズに関する緩やかな条件の下で、理論的な性能保証が与えられている。これは、学習領域全体において高精度を達成する学習アルゴリズムを発見する代わりに、ランダム予測よりほんの少し良い精度を持つアルゴリズムを発見すれば良いことになる。しかし、訓練データが不足ししている場合や複雑すぎる弱仮説が得られる場合、または**BaseLearner**の予測精度が低すぎる場合などは、ブースティングは理論的に効果的ではないとされている。

AdaBoostは様々な研究者によって実験的に評価されている。Freundと Schapire［2］は、**AdaBoost**が決定木を用いた分類システムである**C4.5**、単一ノードからなる決定木（決定株）を用いたアルゴリズムに**AdaBoost**を適用した実験で、AdaBoostを適用した決定株は、**C4.5**と同等の性能を持つことを実証した。また、**C4.5**と組み合わせることにより精度の向上が見られることを示している。

さらに、Quinlan[1]では、**C4.5**にたいして**AdaBoost**、**Bagging**という２つのアンサンブル学習アルゴリズムを低要して実験を行っている。両方のアルゴリズムは、共に**C4.5**に対して有効であるという結果となっている。さらに、**AdaBoost**は、**Bagging**よりも**C4.5**に対して有効であるという結果も示されている。この実験では、ブースティングのトライアル回数は、10回となっている。

他にも様々な学習システムに対する実験でブースティングが有効であるという報告されている。しかし、ブースティングが有効でないケースの報告もあった。たとえばFreundと Schapire[5]によるOCRの実験において、データの中に例外事例が非常に多く存在するとき、困難な事例へ重みを集中させることが学習に重大な悪影響を示すことがあると言う物である。このようなケースに対応するために改良されたアルゴリズムもいくつか発表されている。

また、ブースティングの変わった適用例もあった。それは、例外事例の発見である。ブースティングは、謝った予測をした事例に対して学習を集中させていくという部分を利用した物である。

第 4 章
AdaBoost の改良

4.1 改良の提案

本論文では、**AdaBoost** の動作をより深く分析するために、アルゴリズムの改良を行い、**AdaBoost** との比較を行う。**AdaBoost** の改良は、様々な研究がなされている。たとえば、Quinlan[1]では、最終仮説の方法の改良を行っている。この改良は、そこそこ良い結果が出ていた。また、前章でも紹介したが間違った事例に学習が集中しすぎないように、重み更新の方法を改良した物などがある。

本論文では、重みの更新方法の改良を行った。**AdaBoost** の重み更新の法則は、どのような事例に対しても同じ方法で行われる。しかし、訓練データ内のラベル集合 $y_i \in Y = \{1,\ldots,k\}$ が存在する割合は一定ではなく、存在する割合が少ないラベルは、割合の多いラベルよりも予測するときの重要度は、大きいと考えることが出来る。たとえば、訓練データ数 10 個，ラベル集合 $y_i \in Y = \{1,-1\}$ の訓練データがあるとすると、ラベル1が 7 個あるとすると、ラベル−1は 3 個しか存在しないことになる。このときラベル−1は、ラベル1より予測するときの重要度は大きいと考えられる。このことを **AdaBoost** の重み更新の方法に取り入れて学習を行うアルゴリズムの改良を行った。これによって **AdaBoost** の性能がどのように変化するかを比較することにした。なお、改良したアルゴリズムは、２種類作成した。本論文では、モデル１を **AdaBoost_M1** と呼ぶことにし、モデル２を **AdaBoost_M2** と呼ぶことにする。以下の節でアルゴリズムの説明を行う。

4.2 モデル１（AdaBoost_M1）

このアルゴリズムは、**AdaBoost** の学習プロセスに訓練データ内のラベルの割合による重要度を加えて学習させるために次のような作業を加えた。

まず、学習をする際の準備のときにラベル集合の割合

$$\alpha(y_i) \quad \text{ただし} \quad y \in Y = \{1,\ldots,k\} \text{とする。}$$

を準備する。

BaseLearner による学習、誤り率、仮説の需要どの計算は、通常の **AdaBoost** と同じ方法で行う。

次に、**図４．１**のとおり重みの更新を行う。正しい予測をした物に対しβ_tとその事例の実際のラベルの割合$\alpha(y_i)$をを掛ける。よって、正しい予測をした重みは減少する。さらに、ラベルの割合を掛けることによって、ラベル割合の小さい物の重みはラベルの割合の大きい物の重みよりさらに小さくなる。あとの作業は、通常の **AdaBoost** と同様である。

The algorithm **Modified AdaBoost 1 (AdaBoost_M1)**

Input: m 個の訓練データ $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ ラベル集合は、 $y_i \in Y = \{1, \ldots, k\}$ とする。

ラベル集合の存在する割合：$\alpha(y_i)$

ベースとなる学習機械： **BaseLearner**

トライアル回数 T の設定

Step1: $D_1(i) = 1/m$ によって各事例に対して重みを初期化

Step2: $t = 1, 2, \ldots, T$ に対して

1. 確率分布 D_t を用いて BaseLearner による学習
2. 仮説 h_t: $X \to Y$.を得る。
3. D_t による誤り確率 h_t: $\varepsilon_t = \sum_{i: h_t(x) \neq y_i} D_t(i)$.を計算する。

   もし $\varepsilon_t > 1/2$ ならば、$T = t-1$ として最初のステップに戻る。
4. 仮説の重要度 $\beta_t = \varepsilon_t / (1-\varepsilon_t)$.を計算。
5. 重みの更新 D_t: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t \times \alpha(y_i) & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$

   Z_t は、正規化定数。

Step3: 最終仮説

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}.$$

図４．１：AdaBoost_M1 アルゴリズム

4.3 モデル２（AdaBoost_M2）

このアルゴリズムは、**AdaBoost** の学習プロセスに訓練データ内のラベルの割合による重要度を加えて学習させるためにラベルをわけて重みの更新を行った。イメージは、AdaBoost の学習するスッテプの一つのターンで BaseLearner の学習終了後、訓練データをラベルごとにわけ重みの更新を行いまた、全体を組み合わせるといった方法である。

手順は、**図４．２**より Step2 の２までは通常の AdaBoost と同じである。つぎに、ラベル別の重みを用いた誤り確率

$$\varepsilon(y) \quad \text{ただし} \quad y \in Y = \{1,\ldots,k\} \text{とする。}$$

を計算しする。ここで誤り率が一つでも１/２以上になった場合学習を終了させる。そして、その仮説のラベル別の重要度

$$\beta(y) \quad \text{ただし} \quad y \in Y = \{1,\ldots,k\} \text{とする。}$$

を計算する。つぎにこれらの平均値を求め全体の仮説に対する重要度を求める。重みの更新時は、ラベル別のその仮説に対する重要度を使用して重みの更新を行う。最後に重みの正規化を行う。

最終仮説で使用する仮説別の重要度は Step2 の６で計算した物を使用し多数決を行う。

The algorithm **Modified AdaBoost 1 (AdaBoost_M2)**

Input: m 個の訓練データ $\{(x_1,y_1),...,(x_m,y_m)\}$ ラベル集合は、 $y_i \in Y = \{1,...,k\}$ とする。

ベースとなる学習機械： **BaseLearner**

トライアル回数 T の設定

Step1: $D_1(i) = 1/m$ によって各事例に対して重みを初期化

Step2: $t = 1,2,...,T$ に対して

1. 確率分布 D_t を用いて BaseLearner による学習
2. 仮説 $h_t: X \to Y$.を得る。
3. D_t によるラベル別誤り確率 $\varepsilon(1),..\varepsilon(k)$ を計算する。
   もし $\varepsilon(y) > 1/2$ (ただし $y \in Y = \{1,...,k\}$) ならば、$T = t-1$ として最初のステップに戻る。
4. ラベル別の仮説の重要度を計算 $\beta(y) = \varepsilon(y)/(1-\varepsilon(y))$ ただし $y \in Y = \{1,...,k\}$
5. ラベル別の誤り率の平均を計算 $\varepsilon_t = \sum_{Y=1}^{k} \varepsilon(Y) \Big/ k$
6. 仮説の重要度 $\beta_t = \varepsilon_t/(1-\varepsilon_t)$.を計算。
7. ラベル別の重み更新 $D_t: D'_{t+1}(i) = D_t(i) \times \begin{cases} \beta(y_i) & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
8. 正規化を行う。

Step3: 最終仮説

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}.$$

図４．２：AdaBoost_M2 アルゴリズム

第 5 章
システムの概要

5.1 概要

本研究では、ブースティングと分類システムの正確さとの関係を調べるために以下のようなシステムを作成した。

- C4.5 の枝刈り前の木に AdaBoost を適用した“un-pruned BC4.5”
- C4.5 の枝刈り後の木に AdaBoost を適用した“pruned BC4.5”
- C4.5 の枝刈り前の木に AdaBoost_M1 を適用した“un-pruned BC4.5”
- C4.5 の枝刈り後の木に AdaBoost_M1 を適用した“pruned BC4.5”
- C4.5 の枝刈り前の木に AdaBoost_M2 を適用した“un-pruned BC4.5”
- C4.5 の枝刈り後の木に AdaBoost_M2 を適用した“pruned BC4.5”
- 10-times 10-fold Cross-validation システム “CV”

上記のように 6 種類の分類システム、分類システムを評価するためのシステム、の計７種類のシステムを作成した。また、これらのプログラムは“Microsoft Visual C++ 6.0”で作成した。さらに、C4.5 は、“http://www.cse.unsw.edu.au/~quinlan/”上の“C4.5 Releace8”のソースコードを利用した。なお、分類システムと CV システムの応答は、ファイルを等して行った。

５．２にて分類システム、５．３にて CV システムについての詳しい説明をする。

5.2 分類システム

作成した分類システムは、3種類の Boosting アルゴリズムをそれぞれ C4.5 の枝刈り前後の木に適用した分類システムである。6種類の分類システムを作成したが、基本的にはすべて同じような動作をする。ブースティングアルゴリズムでは、BaseLearner による学習の終了後に、その仮説の重みを用いたエラー率が 0．5を超えた場合は、学習をやり直すことになっているが C4.5 の性質上同じデータセットからは、違う形のルールは作成できないためそこでブースティングの学習をストップさせた。

つぎに C4.5 にブースティングを適用するにための変更点を以下に示す。

- C4.5 の変更点
  - Visual C++でコンパイルできるようにした。
  - 入力を変更した。(以下で説明)
  - C4.5 の学習に必要なパラメータはディフォルト値を使用する。
  - 決定木作成と木の枝刈り以外の機能を削除した。
  - 枝刈り無しの学習も行えるようにした。
  - 出力をテストデータのみのエラー率だけにした。
- ブースティング機能のための追加点
  - ブースティングに必要なパラメータ、関数の追加
  - 試行回数を設定可能にした。
  - 重みを用いたエラー率が０．５を超えた時点で学習を終了させる。
  - ブースティング後のルールに対するテストデータのエラー、試行回数を出力

つぎに、分類システムのエラー率出力までの流れを述べる。

分類システムは、まずファイル名、決定木の種類、ブースティングの試行回数の順に入力を受け取る。ファイル名を“DF“としたときの入力例は、

C:￥BC45 DF　１　１０

のようになる。なお、決定木の種類は、１．Un-pruned tree、２．Pruned tree となっている。入力後プログラムは、“DF.names”、“DF.data”という２種類のファイル

を読み込む。次にパラメータの初期化を行う。
それから以下の作業を試行回数 T 回だけ繰り返す。

1. C4.5 による決定木の生成
2. 重みを用いたエラー率を計算（もしエラー率が０．５以上の場合は学習終了）
3. 重みの更新
4. 重みの正規化

学習終了後、“DF.test”ファイルを読み込む。ブースティングの学習の１回目はすべて AdaBoost が適用されていない分類システム（つまり C4.5）の結果となっているので、そのエラー率を出力する。そして、試行回数分の作成したルールを組み合わせて最終仮説としてそれのエラー率を出力する。出力は、AdaBoost 適用前と適用後の両方のエラー率とブースティングの試行回数の３種類となっている。

作成した分類システムのフローチャートを図５．１に示す。

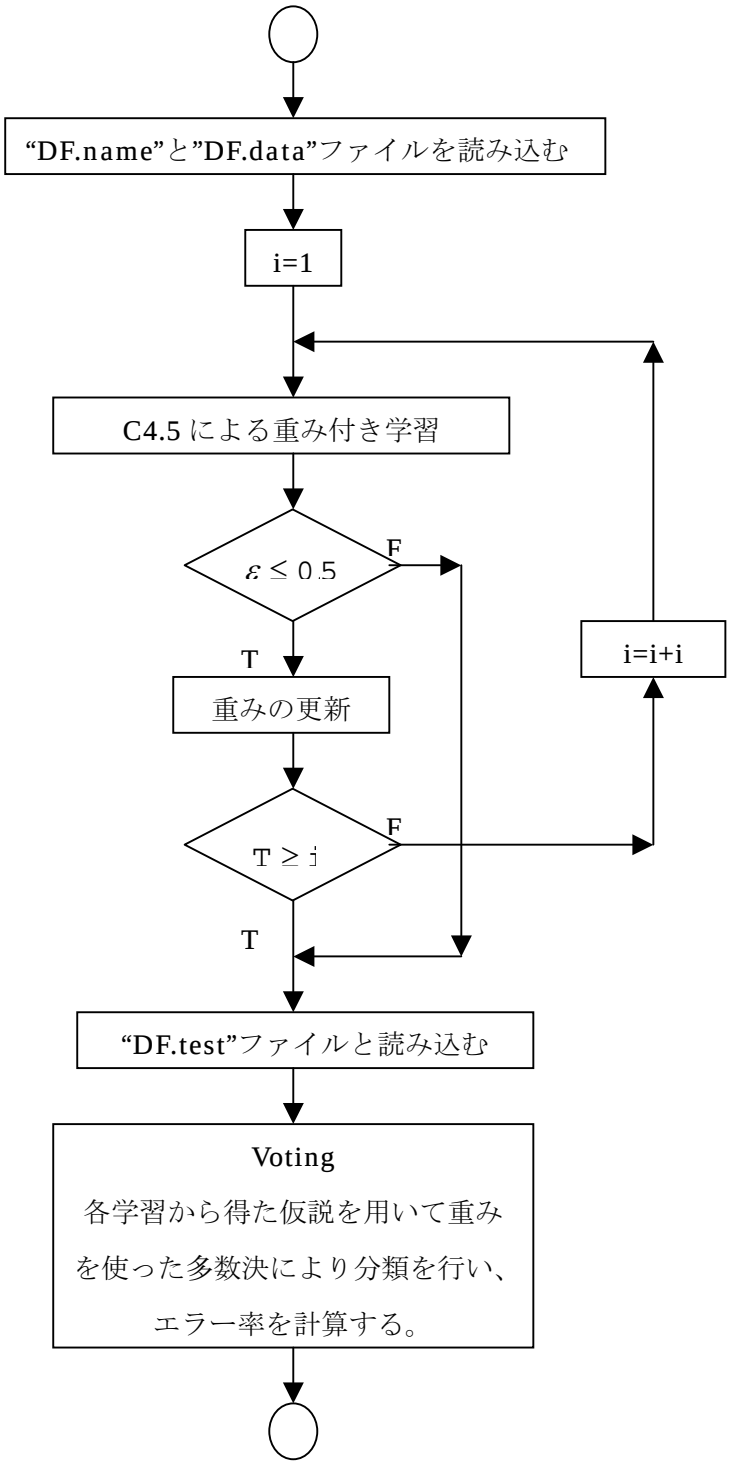


図 5．1 分類システムの簡単なフローチャート

ここでの ε は C4．5 の重みを用いたエラー率、T は試行回数を表している。

5.3 評価システム

CV システムは、分類システムとファイルを利用し応答を行うシステムにした。このシステムの動作は、前節同様データセット名を“DF“とすると、まず“DF.all”ファイルを読み込む。そして以下の作業を 10 回行う。

1. データをシャッフルする。
2. データを 10 個に分ける。
3. 次の作業を分けられたデータそれぞれに対して行う（計 10 回）
   i. 1 つを“DF.test”ファイルに書き込む。
   ii. 残りを“DF.data”ファイルに書き込む。
   iii. 作成した 6 種類の分類システムによる学習を行う。
   iv. 分類システムの出力したエラー率を受け取る。
4. エラー率の平均を計算する。

これらの作業後さらにエラー率の平均を計算しそれを出力する。したがって、このシステムは、分類システムによる分類、計 100 回のエラー率の平均を求めることになる。CV システムの簡単なフローチャートを図 6．2 に示す。

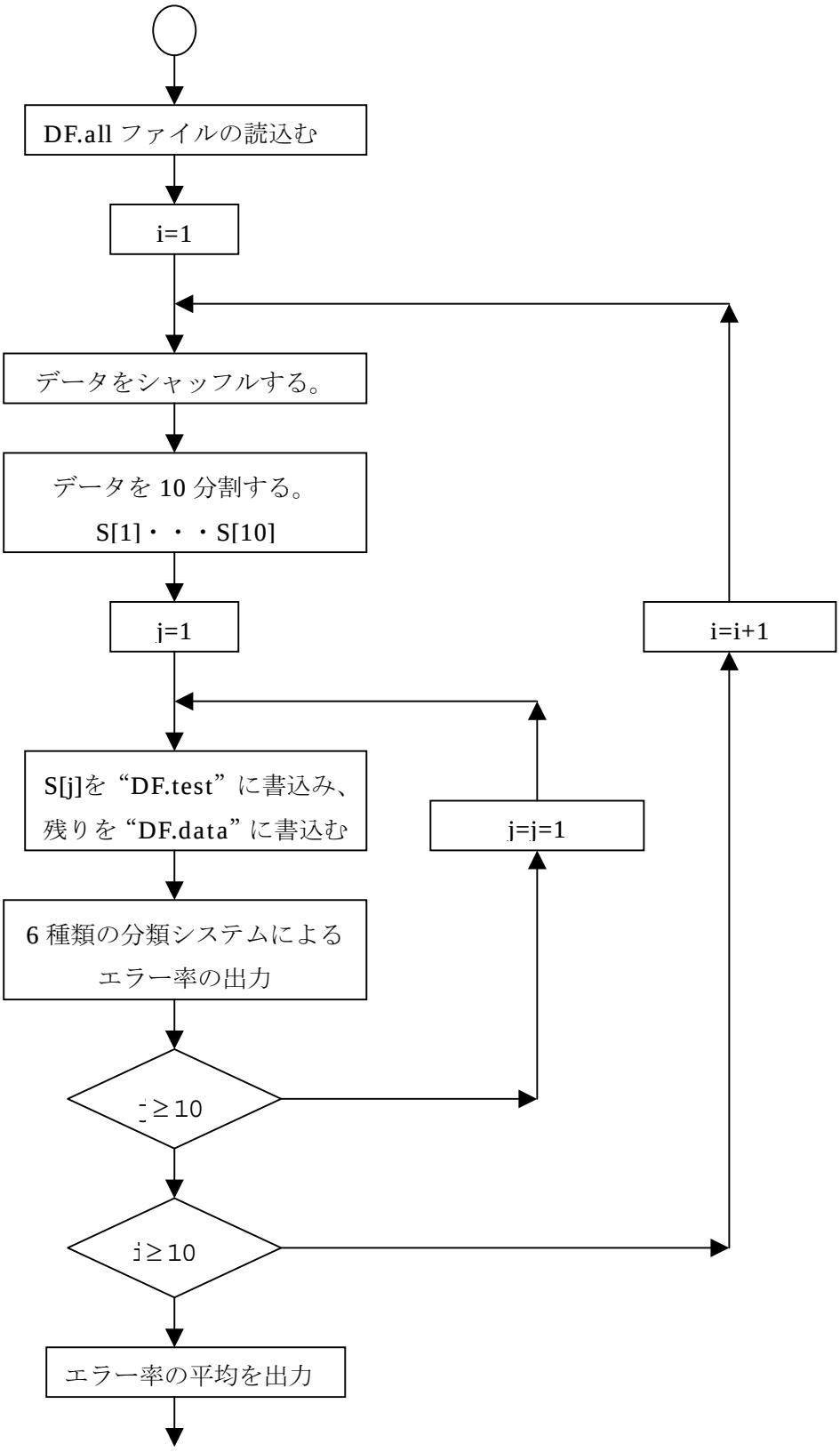

DF.all ファイルの読込む
i=1
データをシャッフルする。
データを 10 分割する。
S[1]・・・S[10]
j=1
S[j]を“DF.test”に書込み、
残りを“DF.data”に書込む
6 種類の分類システムによる
エラー率の出力
j≥10
j=j=1
i≥10
i=i+1
エラー率の平均を出力

図 5．2　評価システムの簡単なフローチャート

第 6 章
実験

6.1 データセット

実験の目的は分類システムの正確さを測ることである。分類システムはデータセットの違いによって異なった正確さを見せる。従ってデータセットの数は多いほどよい。そして種類も豊富である必要がある。また、システムを評価するには奇抜なデータセットは必要なく、よく知られたデータセットを用いることが好ましいとされている。これらの条件をすべて満たす Web 上のデータベースがある。それは、UCI（University of California Irvine）のホームページ (http://www.ics.uci.edu/mlearn/MLRepository.html) 上にあり、学習システムの評価を行う専門家のために維持されている。

本研究では、［1］で用いられている 27 種類のデータセットを中心に、UCI のデータベースから入手した 30 種類のデータセットを使用した。なお、本研究で扱うすべてのシステムは C4．5 の形式によるファイルしか受け付けないのでデータセットはこの形式になるようにした。また、入手したデータセットは欠損値の無い物を選択した。さらに入手したデータセットは、属性値は連続値か離散値となっている。また、クラスは離散値となっている。

表 6．1 に個々のデータセットの特徴をしめした。

データセット名	データ数	クラス数	属性数	
			連続値	離散値
anneal	898	6	6	32
audiology	226	24	70	0
auto	205	7	14	12
breast	699	2	10	0
chess	3500	2	0	36
cleve	303	2	6	7
diabetes	768	2	8	0
DNA	3186	3	0	180
flare	1066	2	2	8
glass	214	5	10	0
heart	270	2	13	0
hepatitis	155	2	6	14
horse-colic	368	2	6	16
hypothyroid	2800	2	7	22
ionosphere	351	2	34	0
iris	150	3	4	0
labor-neg	85	2	8	8
letter	20000	26	16	0
lymphography	148	4	3	15
segment	2310	7	19	0
shuttle	58000	7	9	0
sick-euthyroid	3163	2	7	18
sonar	208	2	59	0
soybean-large	316	19	0	35
splice	3190	3	0	59
vehicle	840	4	18	0
vote	435	2	0	16
waveform-21	5000	3	21	0
wine	178	3	13	0
zoo	101	7	0	18

表６．１　データセットの特徴

6.2 実験の概要

本研究における実験は、

- C4.5 と BC4.5 との比較
- BC4.5 と BC4.5_M1 との比較
- BC4.5 と BC4.5_M2 との比較

これら 3 種類の実験を行った。なお各実験とも un-pruned tree と pruned tree の比較も行っている。

本研究におけるブースティング手法を適用した分類システムのブースティング学習時の試行回数は、[1]の論文同様 10 回として行った。なお、各実験における比較は CV システムを用いることによる推定エラー率を測ることで行った。

6.3 実験の手順

実験の手順を以下に示す。

1. データセットを選択する。
2. データセットを CV システムに読み込ませる。
3. すべての分類システムのエラー率を得る。

この作業をすべてのデータセットに対して行った。すべてのエラー率を出力後上記の 3 種類の比較を行った。

6.4 実験の結果と考察

6.4.1 C4.5 と BC4.5 との比較

	C4.5		BC4.5			
	un-pruned	pruned	un-pruned		pruned	
	err	err	T	err	T	err
anneal	0.0578	0.0793	10.0	0.0508	10.0	0.0569
audiology	0.2296	0.2213	9.9	0.2230	10.0	0.2551
auto	0.1762	0.1962	10.0	0.1640	10.0	0.1821
breast	0.0592	0.0557	10.0	0.0392	10.0	0.0388
chess	0.0056	0.0057	10.0	0.0056	10.0	0.0074
cleve	0.2356	0.2438	10.0	0.1990	10.0	0.2064
diabetes	0.2593	0.2600	10.0	0.2517	10.0	0.2486
DNA	0.0835	0.0761	10.0	0.0832	10.0	0.1744
flare	0.1884	0.1728	9.9	0.1769	10.0	0.1708
glass	0.3169	0.3169	10.0	0.2541	10.0	0.2509
heart	0.2478	0.2178	10.0	0.1996	10.0	0.1915
hepatitis	0.2133	0.2043	10.0	0.1796	10.0	0.1808
horse-colic	0.1742	0.1459	10.0	0.1601	10.0	0.1513
hypothyroid	0.0078	0.0073	10.0	0.0087	10.0	0.0080
ionosphere	0.1075	0.1059	10.0	0.0730	10.0	0.0718
iris	0.0500	0.0580	9.7	0.0553	10.0	0.0600
labor-neg	0.2180	0.2193	9.8	0.1570	10.0	0.1977
letter	0.1190	0.1197	10.0	0.0531	10.0	0.0552
lymphography	0.2601	0.2348	10.0	0.1992	10.0	0.2092
segment	0.0339	0.0333	10.0	0.0206	10.0	0.0215
shuttle	0.0002	0.0003	9.2	0.0002	9.9	0.0002
sick-euthyroid	0.0238	0.0213	10.0	0.0226	10.0	0.0219
sonar	0.2725	0.2592	9.9	0.1971	9.9	0.1816
soybean-large	0.0918	0.0845	10.0	0.0729	10.0	0.0594
splice	0.0789	0.0576	9.9	0.0677	10.0	0.0625
vehicle	0.2789	0.2726	10.0	0.2457	10.0	0.2434
vote	0.0545	0.0476	10.0	0.0517	10.0	0.0435
waveform-21	0.2376	0.2355	10.0	0.1721	10.0	0.1737
wine	0.0711	0.0732	8.6	0.0502	9.4	0.0440
zoo	0.0731	0.0813	2.3	0.0691	8.6	0.1180
ave	0.1409	0.1369	9.6	0.1168	9.9	0.1229

表６．２　C4.5 と BC4.5 との比較

BC4.5 の T はブースティングの試行回数の平均である。

表 6．2 より、C4.5 に関してはその性能通り un-pruned tree よりも pruned tree の方がよい結果となったため、C4.5 の Pruned tree と 2 種類の BC4.5 の比較を行った。

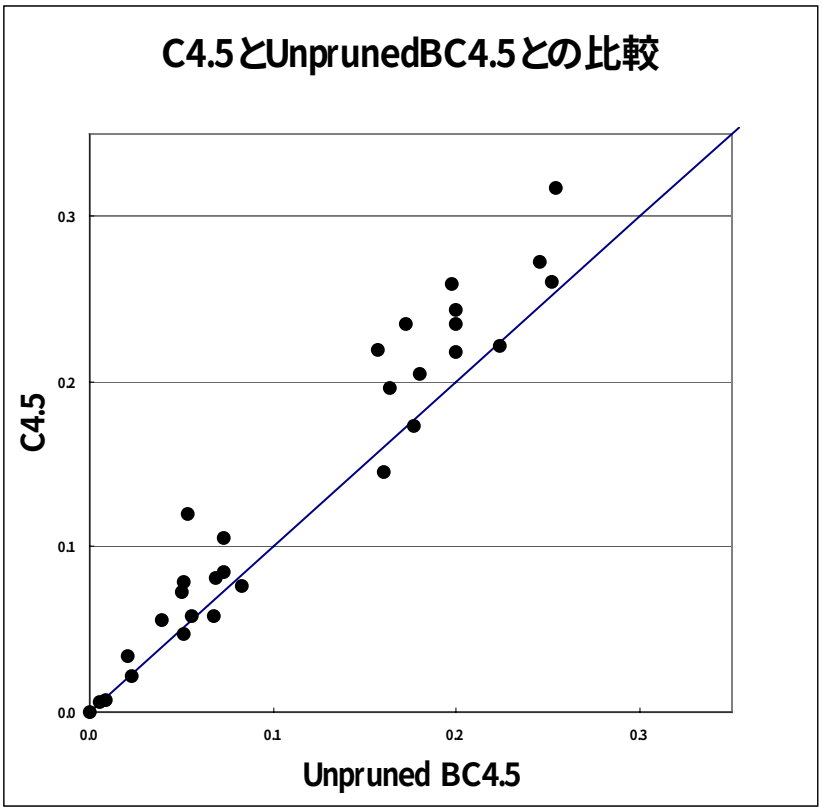


図６．１　Pruned C4.5 と Un-pruned BC4.5 との比較

Un-pruned BC4.5 に関しては、C4.5 の pruned tree に対して 30 個中 23 個のデータでエラー率の減少が見られた。また、平均値を比べると約 2.0%のエラー率が減少みられた。図６．１のグラフは、x 軸に Un-pruned BC4.5 のエラー率をとり、y 軸に Pruned C4.5 のエラー率を取ったグラフである。Un-pruned BC4.5 は 7 個のデータでエラー率の悪化があったが、このグラフをより大幅な悪化はない事が確認された。

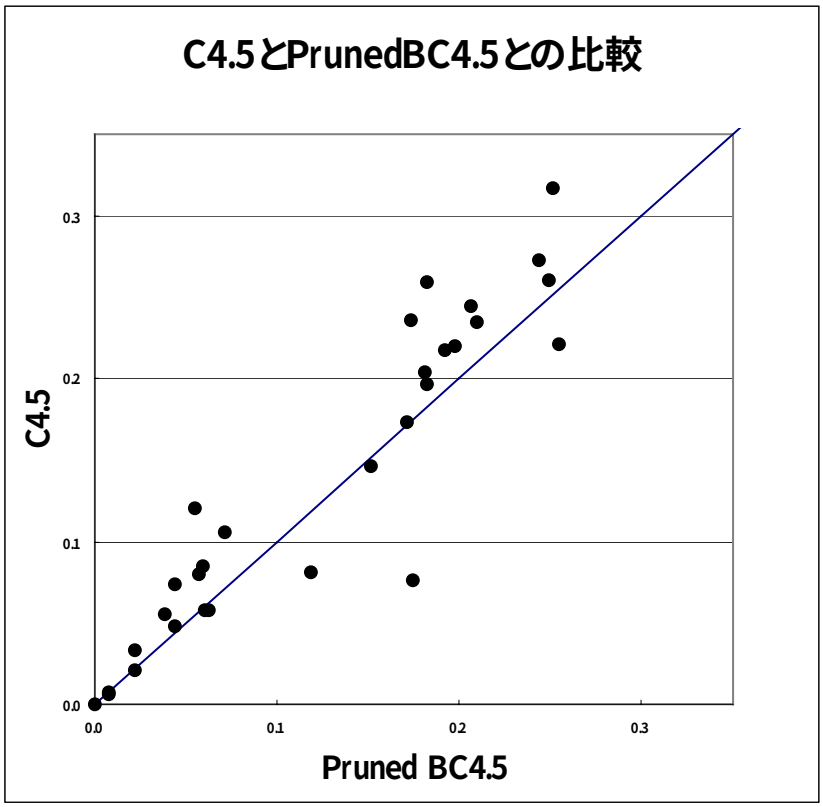


図 6．2　Pruned C4.5 と Pruned BC4.5 との比較

Pruned BC4.5 に関しては、C4.5 の pruned tree に対して 30 個中２４個のデータでエラー率の減少が見られた。また、平均値を比べると約 1.5%のエラー率の減少が見られた。図６．２のグラフは、x 軸に Pruned BC4.5 のエラー率をとり、y 軸に Pruned C4.5 のエラー率を取ったグラフである。Pruned BC4.5 は 6 個のデータでエラー率の悪化があったが、このグラフをよりエラー率が大幅に増大しているデータが 3 種類（audiology, DNA, zoo）確認された。

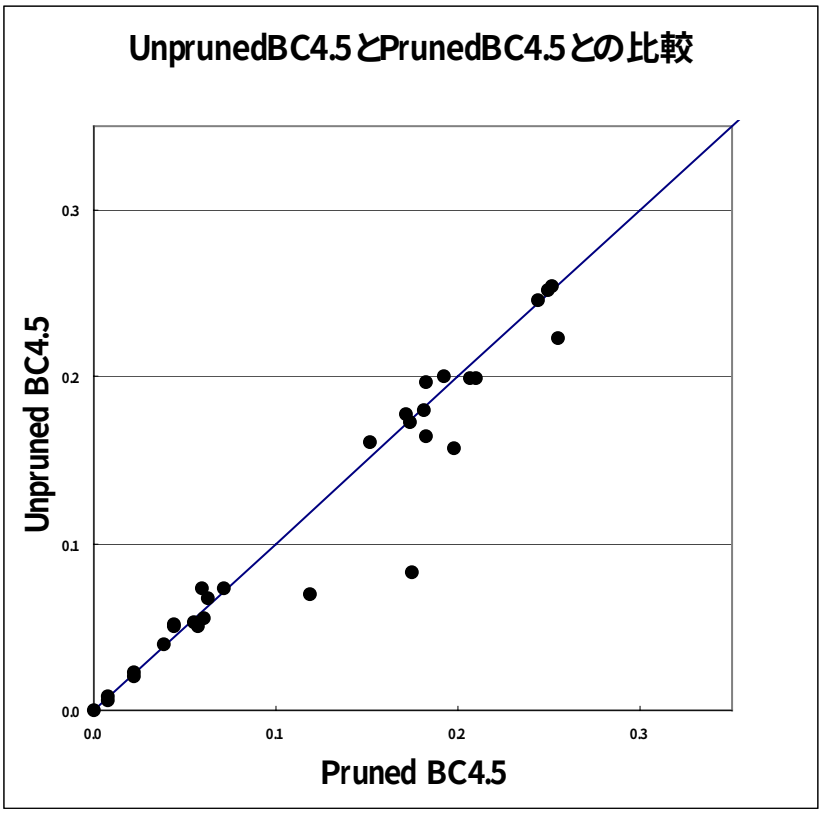


図６．３　BC4.5 の Un-pruned と Pruned の比較

BC4.5 の Un-pruned と Pruned の両方とも C4.5 よりも良い結果となった。平均のエラー率の減少を見ると AdaBoost は、C4.5 の Pruned tree よりも Un-pruned tree へ適用した方が良い結果が得られることが分る。図 6．3 より、BC4.5 の Un-pruned と Pruned では、Pruned BC4.5 が Un-pruned BC4.5 よりも良い結果が得られているデータも多数あるが、数種類のデータでは Un-pruned BC4.5 よりもエラー率が大幅に増大しているデータが見受けられた。これらのことから総合的に AdaBoost は、C4.5 の Pruned tree よりも Un-pruned tree に適用した方が効果的あることが分る。

6.4.2　BC4.5 と BC4.5_M1 との比較

前節で BC4.5 の Un-pruned と Pruned では、Un-pruned の方が良い結果が出ているので、BC4.5_M1 と Un-pruned BC4.5 を比較することにした。

	BC4.5		BC4.5_M1			
	Un- pruned		Un- pruned		Pruned	
	T	err	T	err	T	err
anneal	10.0	0.0508	6.7	0.0597	10.0	0.1173
audiology	9.9	0.2230	1.1	0.2305	7.4	0.2612
auto	10.0	0.1640	10.0	0.1912	9.8	0.2468
breast	10.0	0.0392	9.0	0.0416	10.0	0.0392
chess	10.0	0.0056	10.0	0.0056	10.0	0.0057
cleve	10.0	0.1990	10.0	0.1983	10.0	0.1962
diabetes	10.0	0.2517	10.0	0.2530	10.0	0.2462
DNA	10.0	0.0832	5.7	0.0838	10.0	0.1579
flare	9.9	0.1769	1.0	0.1884	10.0	0.1708
glass	10.0	0.2541	9.3	0.2803	10.0	0.2783
heart	10.0	0.1996	10.0	0.2004	10.0	0.1848
hepatitis	10.0	0.1796	5.4	0.2027	10.0	0.2053
horse- colic	10.0	0.1601	9.9	0.1601	10.0	0.1434
hypothyroid	10.0	0.0087	4.6	0.0083	10.0	0.0165
ionosphere	10.0	0.0730	10.0	0.0778	10.0	0.0794
iris	9.7	0.0553	9.9	0.0573	10.0	0.0633
labor- neg	9.8	0.1570	9.9	0.1720	10.0	0.2280
letter	10.0	0.0531	10.0	0.0635	10.0	0.0639
lymphography	10.0	0.1992	7.7	0.2297	10.0	0.2373
segment	10.0	0.0206	10.0	0.0239	10.0	0.0248
shuttle	9.2	0.0002	9.7	0.0003	10.0	0.0010
sick- euthyroid	10.0	0.0226	3.2	0.0232	10.0	0.0329
sonar	9.9	0.1971	9.9	0.2005	10.0	0.1937
soybean- large	10.0	0.0729	8.9	0.0807	10.0	0.0791
splice	9.9	0.0677	9.3	0.0750	10.0	0.0810
vehicle	10.0	0.2457	10.0	0.2461	10.0	0.2420
vote	10.0	0.0517	10.0	0.0545	10.0	0.0440
waveform- 21	10.0	0.1721	10.0	0.1783	10.0	0.1764
wine	8.6	0.0502	9.8	0.0489	9.8	0.0512
zoo	2.3	0.0691	2.0	0.0788	5.1	0.1447
ave	9.6	0.1168	8.1	0.1238	9.7	0.1337

表 6．3　BC4.5 と BC4.5_M1 との比較

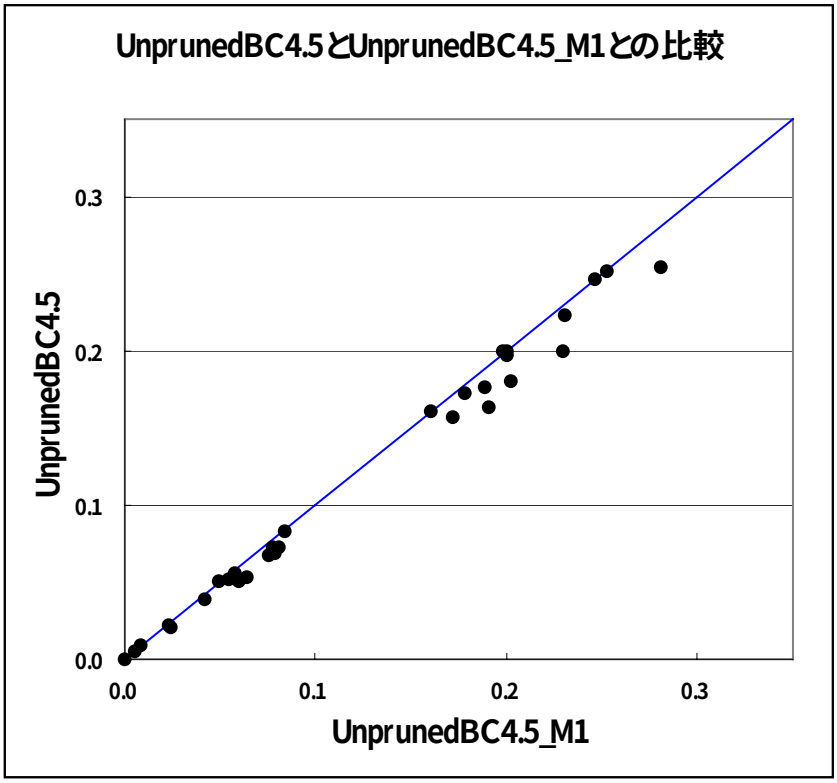


図 6．4　Un-pruned BC4.5 と Un-pruned BC4.5_M1 との比較

Un-pruned BC4.5_M1 は、C4.5 よりはエラー率は良くなっている。しかし、Un-pruned BC4.5 に対して 30 個中 27 個のデータでエラー率の増加が見られ、平均値を比べると約 0.8%のエラー率が増加となっている。図６．４のグラフは、x 軸に Un-pruned BC4.5_M1 のエラー率をとり、y 軸に Un-pruned BC4.5 のエラー率を取ったグラフである。グラフより、Un-pruned BC4.5_M1 は特に大きな悪化は見られなかったが、ほぼすべてのデータに対してエラー率が増加してしまった。

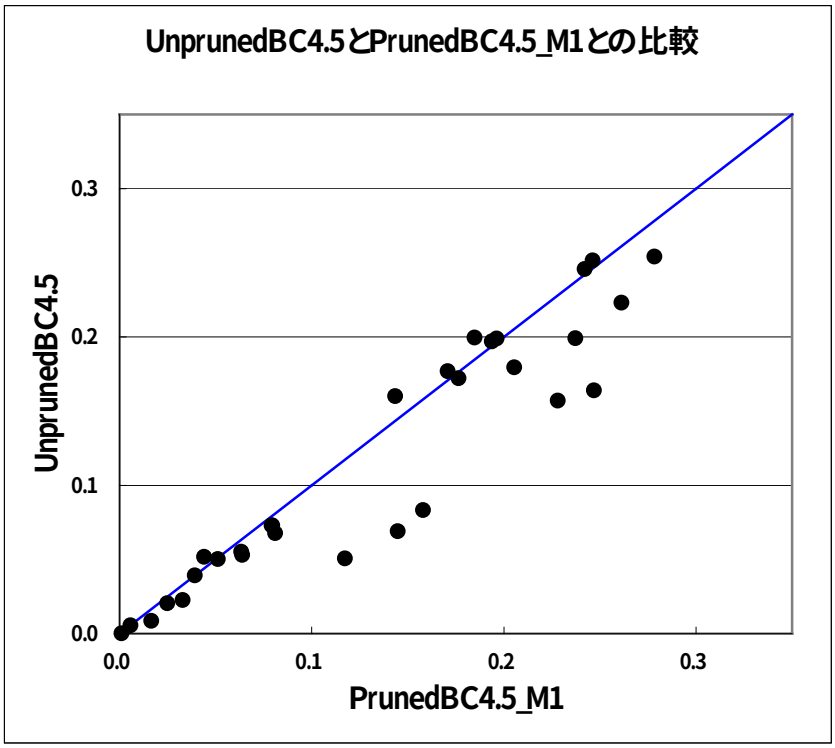


図 6．5　Un-prunedBC4.5 と PrunedBC4.5_M1 との比較

Pruned BC4.5_M1 も、C4.5 よりはエラー率は良くなっている。しかし、Un-pruned BC4.5 に対して 30 個中 7 個のデータでエラー率の減少が見られ、Un-pruned BC4.5_M1 よりはエラー率が減少しているデータは多い事が確認された。しかし、平均値を比べると Un-pruned BC4.5 と比べエラー率が約 1.7%も増加している。図６．5 のグラフは、x 軸に Pruned BC4._M1 のエラー率をとり、y 軸に Un-pruned BC4.5 のエラー率を取ったグラフである。グラフより、Pruned BC4.5_M1 は、エラー率が大きく悪化しているデータがめだっている。全体的には、あまり良くない結果となったが現在比較した分流システム（6 種類）のなかで 4 つのデータに関してエラー率が最小になっているデータもみられた。

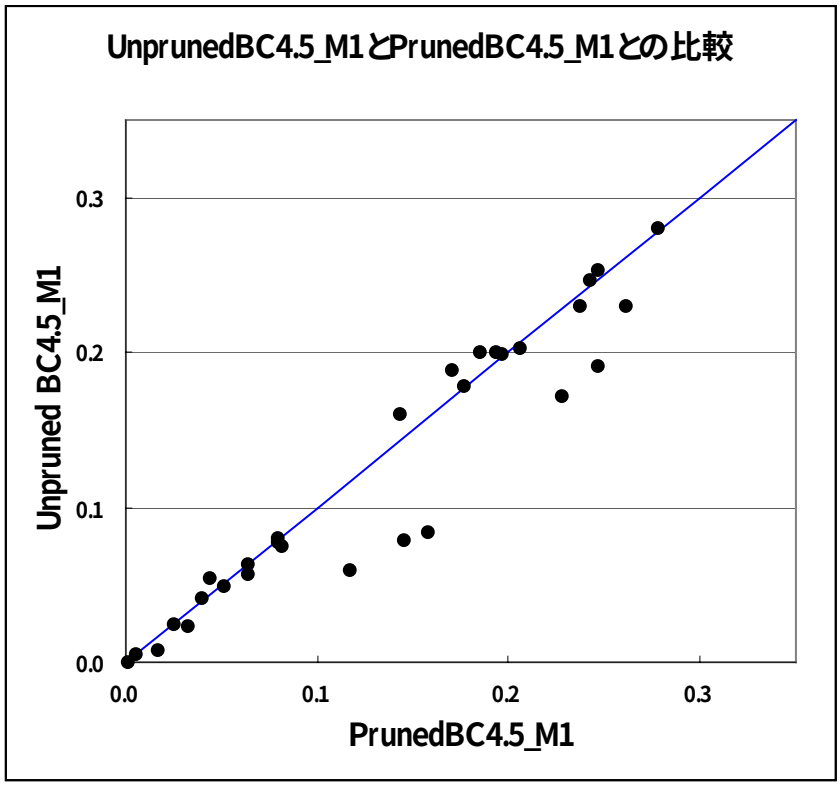


図６．６　Un-prunedBC4.5_M1 と PrunedBC4.5_M1 との比較

BC4.5_M1 の Un-pruned と Pruned の両方とも C4.5 よりも良い結果となった。しかし、両方とも BC4.5 よりも悪い結果となった。図 6．６より、BC4.5_M1 の Un-pruned と Pruned では、Pruned BC4.5_M1 が Un-pruned BC4.5_M1 よりも良い結果が得られているデータは多数あるが、数種類のデータでは Un-pruned BC4.5_M1 よりもエラー率が大幅に増大した。これらのことから総合的に AdaBoost_M1 は、C4.5 の Pruned tree よりも Un-pruned tree に適用した方が効果的であることが分る。

6.4.3　BC4.5 と BC4.5_M2 との比較

前節同様、BC4.5_M2 と Un-pruned BC4.5 を比較する。

	BC4.5		BC4.5_M2			
	Un- pruned		Un- pruned		Pruned	
	T	err	T	err	T	err
anneal	10.0	0.0508	0.9	0.0578	0.9	0.0793
audiology	9.9	0.2230	0.0	0.2296	0.0	0.2213
auto	10.0	0.1640	1.0	0.1762	1.0	0.1962
breast	10.0	0.0392	10.0	0.0382	10.0	0.0373
chess	10.0	0.0056	10.0	0.0058	10.0	0.0170
cleve	10.0	0.1990	10.0	0.2428	10.0	0.2552
diabetes	10.0	0.2517	1.8	0.2591	1.9	0.2602
DNA	10.0	0.0832	10.0	0.0742	10.0	0.0703
flare	9.9	0.1769	0.0	0.1884	0.0	0.1728
glass	10.0	0.2541	1.0	0.3169	1.0	0.3169
heart	10.0	0.1996	10.0	0.2100	10.0	0.1822
hepatitis	10.0	0.1796	7.9	0.1853	2.0	0.2003
horse- colic	10.0	0.1601	10.0	0.1573	10.0	0.1540
hypothyroid	10.0	0.0087	10.0	0.0138	10.0	0.0097
ionosphere	10.0	0.0730	10.0	0.0813	10.0	0.0792
iris	9.7	0.0553	10.0	0.0500	10.0	0.0567
labor- neg	9.8	0.1570	5.4	0.1720	1.7	0.2283
letter	10.0	0.0531	5.6	0.1106	9.1	0.1098
lymphography	10.0	0.1992	1.2	0.2601	0.9	0.2348
segment	10.0	0.0206	10.0	0.0290	9.9	0.0300
shuttle	9.2	0.0002	1.0	0.0002	1.2	0.0003
sick- euthyroid	10.0	0.0226	9.9	0.0298	10.0	0.0256
sonar	9.9	0.1971	6.3	0.2480	7.7	0.2385
soybean- large	10.0	0.0729	0.8	0.0918	0.8	0.0845
splice	9.9	0.0677	10.0	0.0541	10.0	0.0639
vehicle	10.0	0.2457	1.1	0.2789	1.2	0.2726
vote	10.0	0.0517	10.0	0.0446	10.0	0.0437
waveform- 21	10.0	0.1721	10.0	0.1857	10.0	0.1858
wine	8.6	0.0502	10.0	0.0591	10.0	0.0617
zoo	2.3	0.0691	1.0	0.0731	1.0	0.0813
ave	9.6	0.1168	6.2	0.1308	6.0	0.1323

表 6．4　BC4.5 と BC4.5_M2 との比較

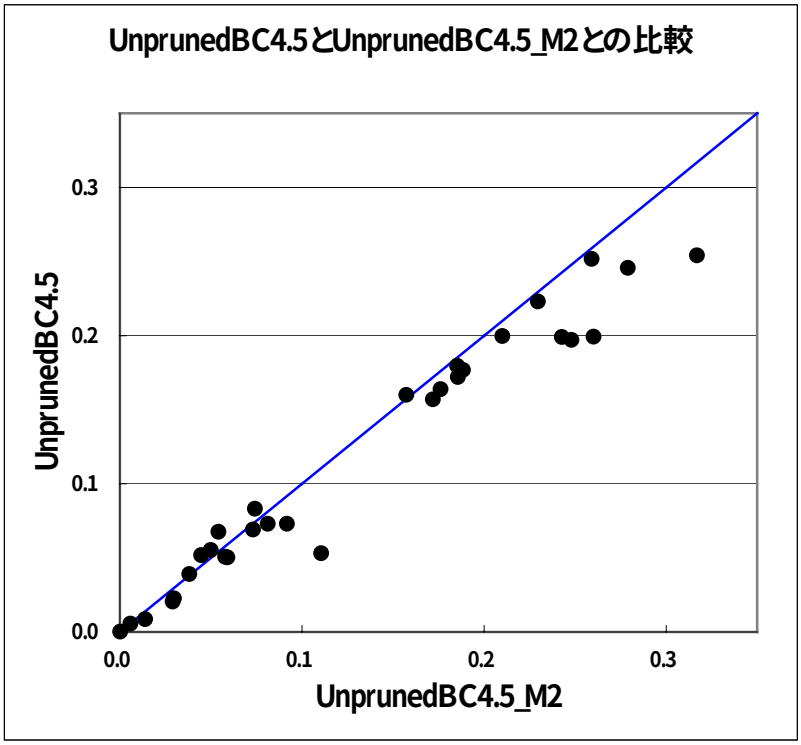


図 6．7　Un-prunedBC4.5 と Un-prunedBC4.5_M2 との比較

Un-pruned BC4.5_M2 は、C4.5 よりはエラー率は良くなっている。しかし、Un-pruned BC4.5 に対して 30 個中 4 個のデータでエラー率の減少が見られるが、平均値を比べると約 1.4%のエラー率が増加となっている。また、ブースティングの試行回数も途中で止まってしまい、学習が止まってしまうデータが多数存在していた。また、図６．７のグラフは、x 軸に Un-pruned BC4.5_M2 のエラー率をとり、y 軸に Un-pruned BC4.5 のエラー率を取ったグラフである。グラフより、Un-pruned BC4.5_M1 はえら率の大きな悪化が見みられ、さらにほぼすべてのデータに対してエラー率が悪化している。

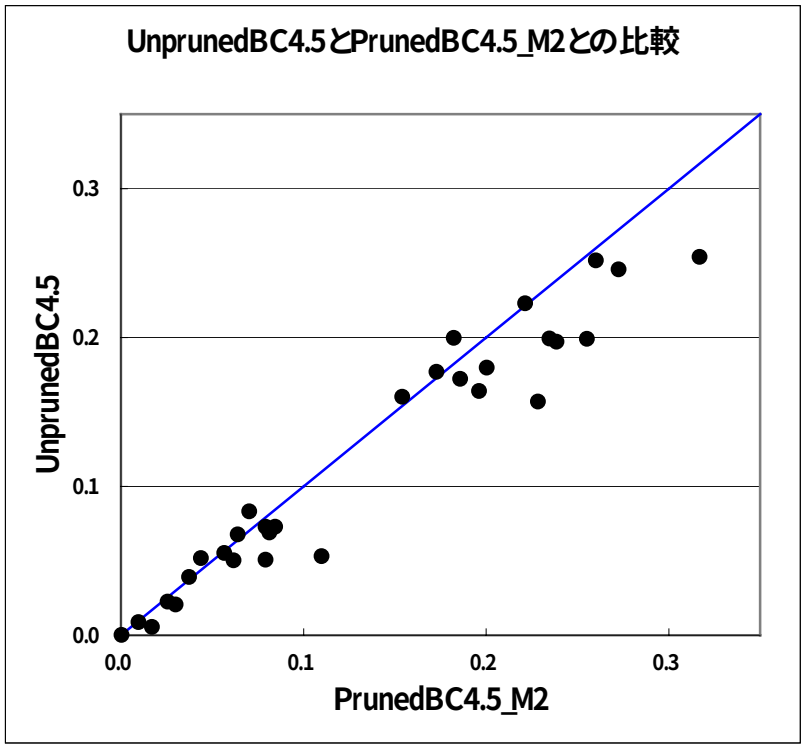


図 6．8　Un-prunedBC4.5 と PrunedBC4.5_M2 との比較

Un-pruned BC4.5_M2 は、C4.5 よりはエラー率は良くなっている。しかし、Un-pruned BC4.5 に対して 30 個中 6 個のデータでエラー率の減少が見られるものの、平均値を比べると約 1.6%のエラー率が増加している。しかし、エラー率の減少したデータ中の 4 個のデータでは実験で用いた分類システムの中でもっともエラー率が低くなった。さらに、2 個のデータでは AdaBoost でエラー率が増大してしまったデータに対しても C4.5 よりエラー率の減少が見られた。また、ブースティングの試行回数も途中で止まってしまい、学習が止まってしまうデータが多数存在していた。図 6．7 のグラフは、x 軸 Pruned BC4.5_M2 のエラー率をとり、y 軸に Un-pruned BC4.5 のエラー率を取ったグラフである。グラフより、Un-pruned BC4.5_M1 はえ

ら率の大きな悪化が見みられ、さらにほぼすべてのデータに対してエラー率が悪化している。

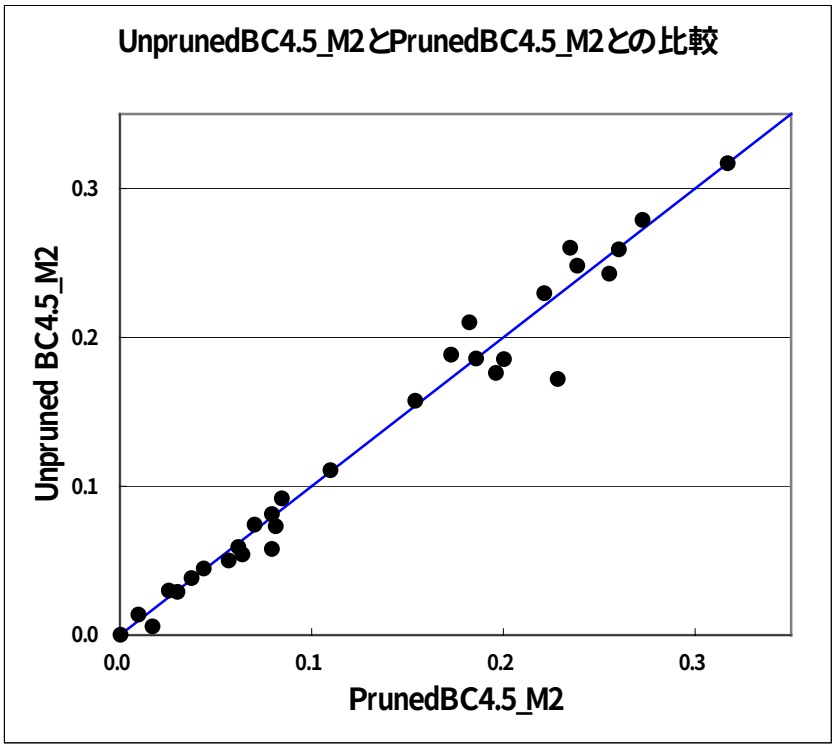


図 6．9　Un-prunedBC4.5_M2 と PrunedBC4.5 との比較

AdaBoost_M2 は、C4.5 のエラー率を減少させることが出来た。しかし、どちらも AdaBoost よりは、悪い結果となった。AdaBoost_M2 は、Un-pruned tree に適用した分類システムの方がエラー率の平均は良かったが、図 6．9 を見ると 2 つのシステムの差は、あまり見られなかった。これらの分類システムは、どちらもブースティングの学習回数が止まってしまい学習が進まないデータが多く見られた。学習を進めるためにアルゴリズムの改良が必要であると感じた。

第 7 章
結論

実験の結果、AdaBoost および改良した AdaBoost は、C4.5 の精度を向上させることが出来ることが確認された。また、C4.5 の“Pruned tree”よりも“Un-pruned tree”に適用した方が C4.5 のエラー率を減少させられることも確認された。ブースティング手法を“Un-pruned tree”に適用した場合はエラー率が増大してしまうデータセットもあったが極端に精度が落ちることは無かったが、“Pruned tree”に適用した場合は、エラー率が極端に増大してしまうデータセットが数個確認された。“Pruned tree”は“Un-pruned tree”よりも精度の高い仮説である。これらの事より、AdaBoost は精度の高すぎる仮説に対しては有効でないと考えられる。しかし、実験の結果より単にエラー率が低い仮説に有効ではないという意味ではなく、ベースとなる学習アルゴリズムの性能が高い物に対してはブースティング手法があまり有効ではないと言うことである。

本研究では、AdaBoost アルゴリズムの改良も行ったが作成したアルゴリズムは AdaBoost よりも C4.5 の精度を向上させることが出来なかった。しかし、AdaBoost_M2 を“Pruned tree”に適用したシステムはエラーの減少率一番悪かったがデータセットによっては、エラー率の減少が最大になる場合があった。また AdaBoost が効果的ではないデータセットに対して有効である場合があった。別の学習アルゴリズムによる実験、さらに改良を加える事が必要だが、このアルゴリズムは性能の良いアルゴリズム、いわゆる強学習アルゴリズムに対して有効になることが考えられる。

最後に今後の課題として、ブースティング手法はベースとなる学習アルゴリズムによって有効性が変化することが確認されたが、データセットによってエラーの減少率の差が多いことが確認された。しかし、実験で使用したデータセットでは、事例の数、属性の数、ラベルの数の違いによる明確な有効性の違いは確認することは出来なかった。これらのことより、ブースティングを効果的に改良するためにはデータセットの違いによる有効性の変化の調査にも注目する必要があると感じた。

謝辞

本研究の開始からまとめまで、研究の全面にわたり一貫してご指導、ご助言を頂きました Ho Tu Bao 教授に深くお礼を申し上げます。研究の方針について助言を頂きました石崎雅人助教授に心から感謝いたします。研究を進めるに当たり、初歩的な質問から研究の方法まで暖かいご意見と丁寧なご指導を頂きました Nguyen Dung Trong 助手に心から感謝いたします。

最後に、本研究を進めるに当たり、あらゆる場面において貴重なご意見を頂きました Ho-石崎研究室の皆様に感謝いたします。

参 考 文 献

[1] Quinlan J.R., 1998, Bagging boosting and C4.5.

[2] Robert E. Schapire, 1999, A Brief Introduction to Boosting.

[3] Michael j. Kearns and Umesh V. Vazirani, 1994, An Introduction to Computational Learning Theory.

[4] Yoav Freund, Robert E. Schapire, 1995, a decision-theoretic generalization of on-line learning and an application to boosting.

[5] Yoav Freund, Robert E. Schapire, 1996, Experiments with a new boosting algorithm.

[6] Robert E. Schapire, 1990, The strength of weak learn ability. Machine Learning.

[7] Yoav Freund, Robert Schapire, (訳：安倍直樹), 1999, ブースティング入門 (A Short Introduction to Boosting).

[8] マイケル J.A.ベリー, ゴードン・リノフ著, SAS インスティチュート, 江原淳, 佐藤栄作訳, データマインニング手法 (Data Mining Techniques) 海文堂出版, １９９９年.

[9] J.R.キンラン著, 古川康一監訳, AI によるデータ解析 (Programs for machine learning) トッパン, 1995 年.

[10] H.M.ダイテル, P.J.ダイテル著, 小嶋隆一訳, Computer Science Textbook C 原簿プログラミング, プレインテスホール出版, 1998.

[11] Jiawei Han and Micheline Kamber (著), Data Mining: Concepts and Techniques, The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor Morgan Kaufmann Publishers, 2000 年.