論　文

線形時間学習及び定数時間識別の一パターン識別手法

小林　卓夫[†]　　中川　正樹[†]

A Pattern Classification Method of Linear-Time Learning and Constant-Time Classification

Takao KOBAYASHI[†] and Masaki NAKAGAWA[†]

あらまし　本論文では，学習サンプル数に比例する空間計算量を上限として，学習サンプル数に対して線形オーダの学習時間と定数オーダの認識時間をもつパターン認識手法を提示する．これらの処理時間は理論上最小のオーダである．学習サンプル集合から識別器を構築するパターン識別方法には，これまでに多くの手法が提案されている．しかし，その多くは学習サンプル数に対して線形を超えるオーダの学習時間がかかる．一方，k-NN 法は学習サンプル数に比例した時間で学習が行えるが，学習サンプル数が多くなると識別時間が線形に増大する問題がある．そこで，本論文では，学習と認識の両方の時間計算量を最小化する新しい手法を提案し，考察を与える．公開された手書き数字データベースなどに対する認識実験を行ったところ，同一の識別率の条件で識別速度を比較すると，1-NN 法の 100 から 500 倍以上，また，高速な NN 探索手法である Locality Sensitive Hashing の 5 から 50 倍以上高速に識別できることが確認できた．

キーワード　学習アルゴリズム，認識アルゴリズム，計算量，手書き数字認識，Locality Sensitive Hashing

1. まえがき

パターン認識では一般に学習サンプル数が多いほど高い識別精度を実現できる．しかしながら，学習サンプル数は，識別精度だけではなく，学習時間，学習に必要とする記憶領域サイズ，識別時間，識別器のサイズなどにも深く関係する．

近年では実用分野におけるパターン識別問題はいっそう大規模化し，大量の学習サンプルを伴う傾向にあり，学習と識別処理の高速化は重要な課題となっている．このような観点から学習と識別処理の複雑さについて既存手法を考察する．

本論文で扱う学習とは教師あり学習を指す．Duda ら [1] にあるように本論文では学習を広義の意味でとらえ，学習サンプル集合を用いて識別器を構築するものを “学習” と呼ぶことにする．

Perceptron をはじめとし，何らかの学習モデルを設定し，ある規準を用いて最適なパラメータやプロトタイプを求めるための様々な手法が提案されてきた．すなわち，Neural Network，Support Vector Machine（SVM）[2]，Learning Vector Quantization（LVQ）[3] などである．これらは多くの場合，識別関数の形はあらかじめ設定されており，最適な識別関数のパラメータやプロトタイプを学習により求めている．

これらはいずれも降下法を用いて，$\Theta(t+1) = \Theta(t) - \alpha_t \nabla E$ という形式で，パラメータの逐次更新を行う．ここで，Θ はパラメータ集合，E は損失，α は学習係数である．基本的に 1 回のパラメータ更新で学習サンプルすべてを参照して損失関数の計算を行い，これを学習が収束するまで繰り返す．そのため，一般に学習の計算量が膨大になるという問題があり，学習サンプル数が大きくなるほど，実用的な適用が困難になる．また，学習を完了したあとで学習サンプルを追加する場合は，一般に学習をすべてやり直す必要がある．

Decision Tree は，識別には木構造探索を行うので，原理的に識別器のサイズが非常に大きくなったとしても高速に識別が行える．しかし，この手法は 1 位候補を返すだけで類似度に相当する情報が得られない欠点がある．学習には，学習サンプルからゆう度が最大になるように分割を繰り返して木構造を構築する手法が

[†] 東京農工大学，小金井市
Tokyo University of Agriculture and Technology, 2–24–16 Naka-cho, Koganei-shi, 184–8588 Japan

提案されている [4], [5].

以上の方法は，いずれも学習に多大な負荷を要する．一方，与えられた学習サンプル集合に対して最も高速に学習できるのが基本的な k-NN 法である [6]．k-NN 法は学習サンプルすべてを記憶領域に格納し，識別器とする．したがって学習はサンプル数に比例した時間で行うことができる．しかしながら，識別時間も学習サンプル数に比例するという欠点がある．

計算幾何学分野における最近傍探索（Closest Point Search, Nearest Neighbor Search：以下 NN 探索）の高速化手法は，直ちにパターン識別手法である k-NN 法の高速化へ適用が可能となるので，いくつかの手法を取り上げる．

Minsky らは理論上の探索時間最小のアルゴリズムを文献 [7] の中で紹介した．これによると，2^b 個の d 次元の 2 値ベクトルの最近傍を得るには，$b \times 2^d$ の記憶領域を使用して，たった 1 度のメモリアクセスで済む．この事実から直ちに，識別時間最速の 2^c 個 $(c \leq b)$ のクラス識別問題が $c \times 2^d$ の記憶領域で実現できることが分かる．しかしながら，この方法は非現実的な記憶領域と学習時間を必要とする．

k-d tree [8] は，代表的な NN 探索のためのアルゴリズムである．しかし，探索時間は次元に対して指数関数的であり，特徴次元数の高いパターン認識に適用するには困難である．また，この手法も最近傍だけを返し，候補情報が得られない．

Kanerva は，データを仮想的に広大なメモリ空間にまばらに分散させて格納・取出しを行う Sparse Distributed Memory（SDM）を提案した [9]．これにより NN 探索を実現できる．しかし，並列処理マシンを前提としており，単一 CPU では処理時間を要する．

以上のように，学習が高速に行える基本的な k-NN 法では識別に時間がかかり，識別関数の形が固定されており比較的識別が速くできる手法では，パラメータ最適化等のために学習に多大な時間がかかるというジレンマがある．特に，識別クラスが多く，それに伴い学習サンプル数が大規模になるほど，学習時間が膨大になる．

本論文ではこれを踏まえて，高速な学習及び高速な識別の両方の特徴をもつ手法 Power Space Similarity method（PSS）を提案する．

PSS は，学習処理は学習サンプル数に対して線形時間で行え，識別処理は学習サンプル数に依存せず定数時間で行える．また，すべてのクラスに対して類似度に相当する尺度を返す．

PSS は，識別処理を完全一致検索処理に落とし込むことにより，学習時及び識別時の空間計算量を多く使用する代わりにこのような特徴を実現する．

PSS は学習・識別に多数の離散的な符号化を行うという部分が Indyk が提案した高速 NN 探索手法である Locality Sensitive Hashing（LSH）[10] に似ている．LSH は，データを複数のハッシュ値で表現してデータの格納・検索を行う．近いデータ同士のハッシュ値は同じ値をとるという原理に基づく確率的なハッシュ関数を使用することにより，NN 探索の近似アルゴリズムを実現する．LSH は，プロトタイプ数にほぼ比例したデータ構造作成時間を実現している．

しかしながら，LSH は NN 探索のためのアルゴリズムで，探索時にクエリとプロトタイプとの厳密な距離計算を伴う．これに対して，PSS はパターン識別アルゴリズムであり，識別関数を作成して各クラスとの類似度を求め，未知パターンと学習サンプルとの間の個々の距離計算は行わない．

Kobayashi は PSS の原型を LSH よりも早く考案し，産業用 OCR 製品に実用化している [11]．しかし [11] では実装方法のみを述べている．本論文では定式化を行い，実験データを示すことにより，提案手法の有効性を明らかにする．

以下，**2.** では PSS の原理を述べ，学習・識別処理における計算量を示す．**3.** では識別性能について理論的な根拠と原理の一般化について考察する．**4.** では手書き数字認識に対する適用実験を示し，1-NN 法と識別性能の比較を行う．**5.** ではパターン識別手法の観点から LSH との性能の比較と考察を行う．また，各種法の学習時間と識別器サイズについて述べる．**6.** ではまとめと今後の課題を述べる．

2. PSS の原理

まず，PSS の基本的な考え方を述べる．

PSS では，特徴ベクトルは離散的な値をとるものとし，これを複数のベクトル（これも離散値のベクトル）に分割する．学習処理では，それらを辞書領域に分散して格納する．識別処理では，入力パターンの特徴ベクトルを学習時と同じ方法で複数ベクトルに分割し，それらをキーとして辞書から一致するベクトルを検索し，そこに記録されているクラス集合の中で最も高い評価関数値を返すクラスを識別結果とする．以下に，原理の詳細を示す．

2.1 表記法の導入

PSS の処理は，様々なベクトル空間とその符号化が関係する．以降の議論のために，いくつかのベクトル空間について特定の表記法を導入する．

N 個の整数，$0, 1, 2, \ldots N-1$ の順列を成分とするベクトルの全体が張る空間を次のように定義する．

$$Facto(N) = \{\boldsymbol{x} \mid \boldsymbol{x} = (k_0, k_1, \ldots, k_{N-1}), \\ k_0 k_1 \ldots k_{N-1} \text{は } 0, 1, 2 \ldots N-1 \text{ の順列 }\} \quad (1)$$

すべての $\boldsymbol{x} \in Facto(N)$ は，大きさが等しく，$|\boldsymbol{x}| = \sqrt{N(N-1)(2N-1)/6}$ である．すなわち，この集合のすべてのベクトルは原点を中心とする N 次元超球面上に乗っている．本論文では，特徴ベクトル空間としてこれを採用する．なお，パターンから $Facto(N)$ に属する特徴ベクトルを取り出す方法は，**5.2** で述べるように，N 次元の実数値特徴ベクトルを作成したあとで，$Facto(N)$ に写像すればよい．

ベクトルの成分として 0 または 1 をとる N 次元ベクトルの全体が作る空間を $\{0,1\}^N$ と表記する．

$$\{0,1\}^N = \{\boldsymbol{b} \mid \boldsymbol{b} = (b_0, b_1, \ldots, b_{N-1}), b_i = 0, 1\} \quad (2)$$

次に，$\{0,1\}^N$ の部分集合として，次の集合を定義する．

$$Combi(N, j) = \{\boldsymbol{e} \mid \boldsymbol{e} \in \{0,1\}^N, |\boldsymbol{e}|^2 = j\} \quad (3)$$

すなわち，成分が 1 のものの個数が j 個，0 のものの個数が $N-j$ 個であるベクトルの全体である．

2.2 ベクトルの分散表現

$Facto(N)$ のベクトルを複数のベクトルに分割する手続きについて述べる．

例として，ベクトル $\boldsymbol{x} = (0, 1, 2, \ldots, N-1)$ を取り上げる．次式は，$\boldsymbol{x}$ を $Combi(N,1), Combi(N,2), \ldots, Combi(N, N-1)$ から一つずつ選んだ $N-1$ 個のベクトルの和で表現している．

$$\begin{aligned}(0, 1, \ldots, N-1) = \quad & (0, 0, \ldots, 0, 0, 0, 1) \\ & + (0, 0, \ldots, 0, 0, 1, 1) \\ & \vdots \\ & + (0, 1, \ldots, 1, 1, 1, 1).\end{aligned}$$

任意の $\boldsymbol{x} \in Facto(N)$ は，同様に次の形で表現できることは自明である．

$$^{\forall}\boldsymbol{x} \in Facto(N), \boldsymbol{x} = \sum_{j=1}^{N-1} \boldsymbol{e}_j, \boldsymbol{e}_j \in Combi(N, j). \quad (4)$$

このとき，上記の $\boldsymbol{e}_j$ の集合 $\{\boldsymbol{e}_j\}$ を，$\boldsymbol{x}$ の分散表現と呼ぶことにする．

2.3 評 価 関 数

今，クラス数を n_C とし，クラス $C_k (k = 1 \sim n_C)$ に属する m_k 個の学習パターンのうち i 番目の学習サンプルを $\boldsymbol{x}^{(i,k)} (i = 1 \sim m_k)$ と書くことにする．全学習サンプル数を m とする $(m = \Sigma m_k)$．また，未知パターンを $\boldsymbol{x}^*$ とする．

パターン $\boldsymbol{x}^{(i,k)}$，$\boldsymbol{x}^*$ の分散表現をそれぞれ $\{\boldsymbol{e}_j^{(i,k)}\}$，$\{\boldsymbol{e}_j^*\}$ とする．このとき，識別のための評価関数を次のように定義する：

$$s_k(\boldsymbol{x}^*) = \left| \{\boldsymbol{e}_j^*\} \cap \bigcup_{i=1}^{m_k} \{\boldsymbol{e}_j^{(i,k)}\} \right|. \ (j = 1 \sim N-1) \quad (5)$$

これは $\{\boldsymbol{e}_j^*\}$ のうち，学習パターン $\{\boldsymbol{e}_j^{(i,k)}\}$ の中に一致するものが存在する個数を示している．クラスごとにこの値を計算し，最大値をとるクラスを識別結果とする．$s_k(\boldsymbol{x}^*)$ は $0 \sim N-1$ の範囲の値をとり，学習に使用したサンプルは必ず最高点 $(N-1)$ となることは自明である．

2.4 学習モデルとしての識別関数

式 (5) の計算は，仮想的に，2^N 次元で入力ベクトルとクラスのベクトルとの内積計算を行っていると考えられる．それを以下に述べる．

空間 $\{0,1\}^N$ のべき集合 $2^{\{0,1\}^N}$ を考えれば，$\{\boldsymbol{e}_j^*\}$，$\bigcup_{i=1}^{m_k} \{\boldsymbol{e}_j^{(i,k)}\}$ のどちらも，$2^{\{0,1\}^N}$ の元である．言い換えれば，2 進数

$$u = u_0\ u_1 \ldots u_{N-1} (u_k = 0 \text{ or } 1, u = 0 \sim 2N-1)$$

を用いて，2^N 次元の 2 値ベクトル $\boldsymbol{b} = (b_u) \in \{0,1\}^{2^N}$ を考えるとき，b_u をオン･オフすることにより，$\{0,1\}^N$ の任意の部分集合を $\{0,1\}^{2^N}$ に属する 1 個のベクトルとして表現することができる．

したがって，$\boldsymbol{e}_j^*$，$\boldsymbol{e}_j^{(i,k)}$ はすべて $\{0,1\}^N$ のベクトルであることに注意すると，$\{\boldsymbol{e}*_j^*\}$，$\bigcup_{i=1}^{m_k} \{\boldsymbol{e}_j^{(i,k)}\}$ を $\{0,1\}^{2^N}$ のベクトル $\boldsymbol{b}^* = (b_u^*)$，$\boldsymbol{b}^k = (b_u^k)$ で表現することができる．このとき，式 (5) は，次の式 (6) に示すように二つのベクトルの内積計算を行うことと等

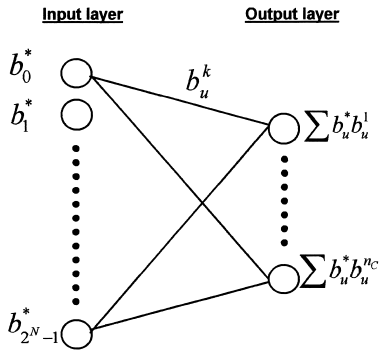


図 1　PSS の線形識別構造
Fig. 1　PSS as a linear classifier structure.

価になる．

$$s_k(\boldsymbol{x}^*) = \sum_{u=0}^{2^N-1} b_u^* b_u^k = (\boldsymbol{b}^*, \boldsymbol{b}^k) \qquad (6)$$

すなわち 2^N 次元の空間で線形識別を行っている．

式 (6) において線形識別パラメータであるそれぞれの b_u^k は，学習サンプルの分散表現で $\boldsymbol{e}_j^{(i,k)} = (u_0, u_1, \ldots, u_{N-1})$ となるものが一つでも現れれば 1 がセットされ，それ以外は 0 となる．これは仮想的に 2^N 次元での識別関数のパラメータ学習を行っていることを意味する．これが PSS の学習モデルである．

図 1 に，識別関数の入力，重み，出力の構成を図示する．

2.5　アルゴリズムの実装と計算量

各クラス $C_k (k = 1 \sim n_C)$ について，式 (5) の値を計算する識別器を作るためには，$\boldsymbol{e}_j^*$ と一致するものを $\{\boldsymbol{e}_j^{(i,k)}\}$ の中から検索できる構造を作ればよい．

追加と検索が高速に行えるデータ構造のためのアルゴリズムとして，ハッシュ法，トリー，ソート済みデータに対するバイナリーサーチなどがある．

ハッシュアルゴリズムのオープンアドレス法では，1 件のデータの追加，検索は総データ件数に対して $O(1)$ 時間で行える．これをもとに学習，識別の時間を見積もると，学習時間の計算量は，$k = 1 \sim n_C$, $i = 1 \sim m_k$, $j = 1 \sim N-1$ について，すべての $\boldsymbol{e}_j^{(i,k)}$ をクラス番号 k とともに識別構造に追加することであるので，$O(Nm)$ となる．

また，識別器の大きさはハッシュ表のサイズとなるが，最終的なロードファクタ（ハッシュ表の利用率）を固定すれば，$\left|\bigcup_{i=1}^{m_k} \{\boldsymbol{e}_j^{(k),i}\}\right|$ が識別器のサイズと考えることができるので，これは $O(Nm)$ を下回る．学習時の空間計算量もこれに等しい．

識別時間の計算量は，初期値 $s_k = 0$ (for all k) とし，$\boldsymbol{e}_j^* (j = 1, \ldots, N-1)$ について，一致するものを識別器の中から検索し，ヒットしたクラス番号 k の s_k の値をインクリメントすればよいので，$O(Nn_C)$ となる．

以上をまとめると，次元数 N，クラス数 n_C を固定すれば，学習は学習サンプル数 m に対して比例した時間で行え，識別は定数時間で行える．したがって，本提案手法は，学習サンプル数に関して基本的な k-NN 法に匹敵する最も速いオーダの学習手法を実現できる．更に，識別についても，最速オーダの識別手法である．

本提案手法では，すべてのクラスに対して評価関数を算出している．つまり，すべてのクラスの類似度情報が得られる．したがって，識別時間はクラス数には比例する．

3.　理論的考察

ここでは PSS の識別性能に関する理論的根拠と原理の一般化を考察する．

3.1　サンプルの分布

学習サンプル集合はパターンの真の出現分布に従い，ランダムにサンプリングされたものとする．

関数 $p_k(\boldsymbol{x})$ を，クラス C_k に関する確率密度関数とする．$Facto(N)$ は離散的な空間なので，学習サンプルを 1 個抽出したとき，$\boldsymbol{x}$ におけるクラス C_k に属するパターンの出現確率を $p_k(\boldsymbol{x})$ とする．

ベイズ識別則により，未知パターン $\boldsymbol{x}$ に対して，確率密度関数最大となるクラスを求めることが最適な識別結果であるので，$k_{max} = argmax\{p_k(\boldsymbol{x})\}$ となるクラス C_{kmax} を見つけることが目的となる．

今，確率密度関数は急激に変化せず，局所的には大体一定であると仮定する．つまり，点 $\boldsymbol{x}$ を含む局所的な領域 $R(\boldsymbol{x})$ では，$^\forall \boldsymbol{x}' \in R(\boldsymbol{x})$，$p_k(\boldsymbol{x}') \cong p_k(\boldsymbol{x})$ であるとする．

3.1.1　シミュレーション（その 1）

式 (5) において $s_k(\boldsymbol{x}^*)$ は $p_k(\boldsymbol{x}^*)$ が変化したときにどのような値をとるか確かめることにする．

すなわち，ある観測点 $\boldsymbol{x}^*$ で $p_k(\boldsymbol{x}^*)$ を変化させたとき，確率密度関数に従う m 個のサンプルをランダムに発生させて，$s_k(\boldsymbol{x}^*)$ がとる値の分布を求める．

これは次のように形を変えて計算機プログラムを作成し，シミュレーションを行うことで実現できる．

空間全体で C_k に属するパターンの出現確率は一様

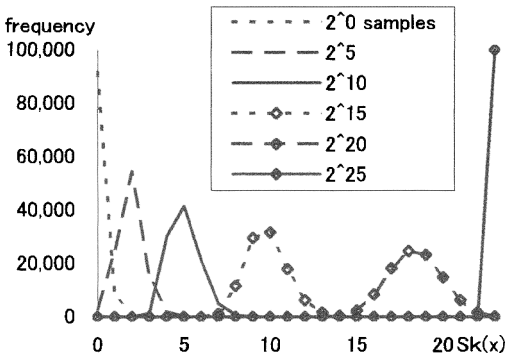


図 2　サンプル数を変化させたときの $s_k(\boldsymbol{x})$ のヒストグラム
Fig. 2　Histogram of $s_k(\boldsymbol{x})$ according to the number of samples.

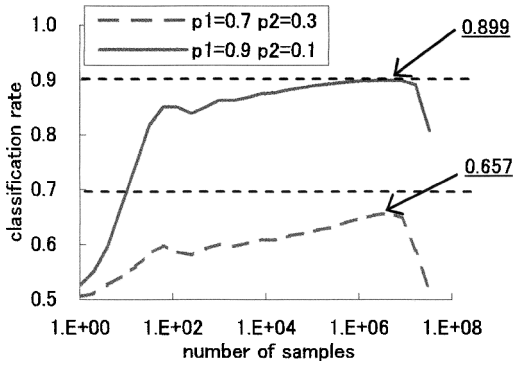


図 3　サンプル密度と認識率の関係（$N = 24$）
Fig. 3　Relation between sample density and classification rate.

であるものとする．C_k に属するサンプルを空間内にランダムに m_k 個発生させる．多数の観測点 $\boldsymbol{x}$ で $s_k(\boldsymbol{x})$ の値を計算し，ヒストグラムをとる．固定した m に対して $p_k(\boldsymbol{x})$ を変化させるということは m_k が変化することであるから，様々に m_k を変化させて計測を行う．シミュレーションでは次元数 $N = 24$ とし，m_k を $2^0 \sim 2^{25}$ とし，観測点を $Facto(N)$ のランダムな 10 万点とした．

図 2 に示されたとおり，$s_k(\boldsymbol{x})$ はサンプル数と対応し，$s_k(\boldsymbol{x})$ の平均値付近を中心にして釣鐘型に分布する．この観察から，ある観測点 $\boldsymbol{x}^*$ における $s_k(\boldsymbol{x}^*)$ にはその点における $p_k(\boldsymbol{x}^*)$ の情報が含まれているといえる．したがって，各クラスについて $s_k(\boldsymbol{x}^*)$ の値を計算し，最大の値をとるクラス C_k を識別結果とすることができると期待できる．

3.1.2　シミュレーション（その 2）

しかし，$s_k(\boldsymbol{x}^*)$ の値にはばらつきがあるので，$p_1(\boldsymbol{x}^*) > p_2(\boldsymbol{x}^*)$ のとき必ずしも $s_1(\boldsymbol{x}^*) > s_2(\boldsymbol{x}^*)$ となるとは限らない．ここでは，学習サンプル数を変化させたときに，$s_k(\boldsymbol{x}^*)$ を測ることによってどの程度の識別能力があるかを確かめるために，次のような計算機シミュレーションを行う．

二つのクラス C_1，C_2 の場合を考える．空間全体で $p_1(\boldsymbol{x})$，$p_2(\boldsymbol{x})$ は一様とし，サンプル数 m を変化させたときの，$s_1(\boldsymbol{x})$，$s_2(\boldsymbol{x})$ を識別尺度としたときの精度を調べる．$p_1 : p_2 = 0.7 : 0.3,\ 0.9 : 0.1$ の場合について実験した．次元数 $N = 24$ とした．なお，$s_1 = s_2$ のときはクラス C_1，C_2 をランダムに選ぶことになる（シミュレーションでは 1/2 ずつ正解のカウントを配分する）．図 3 に結果を示す．

認識率は，おおむね，学習サンプル数を増加させると単調に上昇し，極大に到達したあと下降に転じる．おおむね $m = 2^5 \sim 2^{24}$ と広い範囲にわたり，それぞれの場合で，1-NN 法の理論的最良の認識率である $0.7^2 + 0.3^2 = 0.58$，$0.9^2 + 0.1^2 = 0.82$ を上回っている．最大値はそれぞれ，0.657，0.899 となり，ベイズ識別性能 0.7，0.9 にかなり近いことが分かる．つまり，完全ではないが，確率的にかなりの識別能力があることが分かる．

以上の実験は $p_1(\boldsymbol{x})$，$p_2(\boldsymbol{x})$ が空間内で一様という前提であったが，$p_1(\boldsymbol{x})$，$p_2(\boldsymbol{x})$ が一様でない一般の場合についてここで述べる．この場合も，空間内の各局所領域では確率密度関数は一定と仮定できる．したがって，局所領域の認識率の総和である全体の認識率もサンプル数がある範囲ではほぼ単調に増加すると推測される．このことから，一般の問題への適用においても，学習サンプル数を調節することにより，認識率を極大に到達させることができると思われる．そして，そのときの認識性能が実用的に十分なレベルであれば，このアプローチは有用であるということになる．

3.2　原理の一般化

PSS の原理を **2.**〜**3.1** では特定の空間を使って議論した．しかし，ベクトルを複数のベクトルに分割して表現するという発想は，特定の空間の場合でなくて，もっと一般化することができる．

すなわち，S を特徴ベクトル空間として，$Q_1, Q_2, \ldots, Q_K$ を離散的な空間とするとき，

$$\boldsymbol{x} \in S, \boldsymbol{x} = \sum_{j=1}^{J} \boldsymbol{e}_j, \boldsymbol{e}_j \in Q_k, k = 1, 2, \ldots, K \quad (7)$$

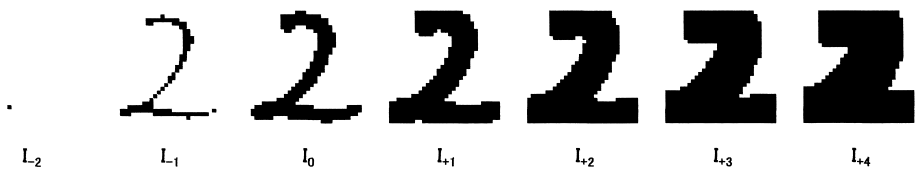
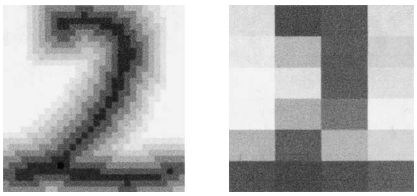


図 4　細め処理と太め処理
Fig. 4　Erosion and dilation processes.

と，一般性を失うことなく $\boldsymbol{x}$ を $\{\boldsymbol{e}_j\}$ に分割する操作が実現できる．ここで $J \geq K$ とする．このようにしても，**2.**〜**3.1** の議論は成り立つ．

例えば，ベクトル $\boldsymbol{x}$ は実数ベクトルであり，大きさがすべて 1 にそろえられているとする．ランダムな直交行列 $M_1, M_2, \ldots, M_K$ を用意し，$\boldsymbol{x}$ に $M_1, M_2, \ldots, M_K$ を掛けて K 個のベクトル $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_K$ を作成する．これらを $\{0,1\}^N$ に量子化したものを $\boldsymbol{x}$ の分散表現とする．これにより実数ベクトル空間でのパターン認識が実現できる．我々はこの方式においても **3.1.2** のシミュレーションを行い，同様の結果が得られることを確認した．



(a)　(b)

図 5　特徴ベクトル $\boldsymbol{A}$ の作成 (a) 図 4 (I_{-2})〜(I_{+4}) を重ね合わせた画像．(b) 作成した特徴ベクトルの視覚化）
Fig. 5　Preparation of the feature vector $\boldsymbol{A}$. (a) superimposed image from images in Fig. 4, (b) visualization of a created feature vector.

4. 手書き数字認識による実験

ここでは，実際のデータを用いて PSS の評価実験を行う．頻繁に再学習することを要したり，カテゴリー数が膨大な識別が本手法の評価には向いているが，公開されたデータベースに対して評価を示すために手書き数字パターンを使用する．

4.1 実験の方法

実験には，公開されているオフライン手書き数字データベース MNIST [12] と，企業により収集された手書き数字データベース（BIRDS database）を使用する．

MNIST は，学習用サンプルパターン数 60,000 個，評価用パターン数 10,000 個で，現在までに多くの手法の性能実験が行われているため，提案手法の性能の位置付けが分かる．しかし，提案手法において，学習サンプル数と認識性能の関係を見るためには MNIST のサンプル数は十分ではない．提案手法では，学習・認識処理が高速であり，60,000 個の学習はすぐに終わってしまい，その学習の本来の能力をもてあます．そこで，BIRDS database を使用し，より大量の学習サンプルを使って，学習サンプル数と認識率の関係を調査する．また，認識時間，使用メモリサイズについて測定する．比較のために，同一の特徴ベクトルを用いた 1-NN 法での性能も測定する．

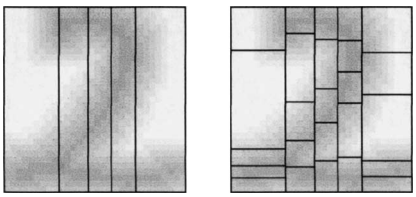
4.2 特徴ベクトル作成方法

特徴ベクトルの作成方法について述べる．MNIST は，手書き文字画像（以下，文字画像）がモノクロ多値データで格納されている．特徴抽出を行うために，前処理として 2 値化処理と文字画像の切出し処理を行う．2 値化の方法は，元画像のピクセル値が 256 階調なので，しきい値 128 で 2 値に変換する．次に，黒画素の XY 座標値の最大最小を求め，文字画像の外接矩形で切出しを行う．

PSS では，特徴ベクトルは $Facto(N)$ に属する必要がある．そこへ到達する前に，まず，2 値の文字画像から N 次元実数ベクトルを作成する．

もとの文字画像を I_0 とする．これに対して，太め処理を段階的に施したもの I_{+1}, I_{+2}, I_{+3}, I_{+4}，細め処理を段階的に施したもの I_{-1}, I_{-2} を作る（図 4）．これらをすべて重ね合わせて，多値画像を作る（図 5 (a)）．この処理は文字の太さに対して正規化を行う効果がある．

こうしてできた画像に対して，領域を縦横 N_X，N_Y に等分して，$N = N_X \times N_Y$ 個の小領域を作る．この小領域内の画素値の合計を成分とするベクトルを作る．これで N 次元実数ベクトルが得られる．これを特徴ベクトル $\boldsymbol{A}$ とする（図 5 (b)）．



(a) (b)

図 6 特徴ベクトル $\boldsymbol{B}$ の作成
Fig. 6 Preparation of the feature vector $\boldsymbol{B}$.

また，別の特徴ベクトル作成方法について述べる．各小領域の画素値の合計が等しくなるような小領域に分割することを考える．そのためには，図 6 (a) に示されているように，まず N'_X 個の縦分割で画素値合計が等しくなるように分割位置を決定する．次にその各領域について，画素値合計が等しくなるように N'_Y 個の横分割位置を決定する（図 6 (b)）．こうして分割された小領域の面積を成分とする特徴ベクトルを作る．これを特徴ベクトル $\boldsymbol{B}$ とする．

こうして得られた実数ベクトルを $Facto(N)$ のベクトルに写像する方法を述べる．この処理はベクトルの量子化である．すなわち，N 次元実数ベクトル $\boldsymbol{v} = (v_0, v_1, \ldots, v_{N-1})$ に最も近い $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1}) \in Facto(N)$ を選択する．これは，ベクトル $\boldsymbol{v}$ の成分の順序情報だけに縮約することであり，$\boldsymbol{v}$ の成分を小さい順に

$$v_{i_0} < v_{i_1} < \cdots < v_{i_{N-1}}$$

と並べたときの序数に着目して，

$$x_{i_0} = 0, x_{i_1} = 1, \cdots, x_{i_{N-1}} = N - 1$$

となるようにベクトル $\boldsymbol{x}$ の成分を設定すればよい．

実験では，特徴ベクトルとしてこの二つの方法を採用し，それぞれ識別器を作成し，二つの認識結果を足し合わせたものを最終的な認識結果とする．

4.3 MNIST を用いた実験

実験では，次元の大きさを 3 段階用意した．すなわち，特徴 A$(N_X \times N_Y)$+ 特徴 B$(N'_X \times N'_Y) = 3 \times 5 + 4 \times 4 = 31$ 次元（small），$4 \times 6 + 5 \times 5 = 49$ 次元（medium），$5 \times 7 + 6 \times 6 = 71$ 次元（large）である．

MNIST の 60,000 個の学習サンプルでは，medium が一番認識率がよかった．比較のため，同一条件による 1-NN 法の測定も行った．実験環境は，PC（CPU：Celeron2.80 GHz，メモリ 1 GByte），OS は Windows-Xp である．測定結果を表 1 に示す．

表 1 MNIST に対する認識性能
Table 1 Classification performance on MNIST.

	PSS	1-NN
認識率	95.94%	96.54%
1 文字当たりの認識時間	0.0494 msec. (20228char./sec.)	8.1659msec. (122char./sec.)
（特徴抽出処理）	(0.0381 msec.)	(0.0381 msec.)
（識別処理）	(0.0113)	(8.1278)
使用メモリサイズ	11MB	3.18MB

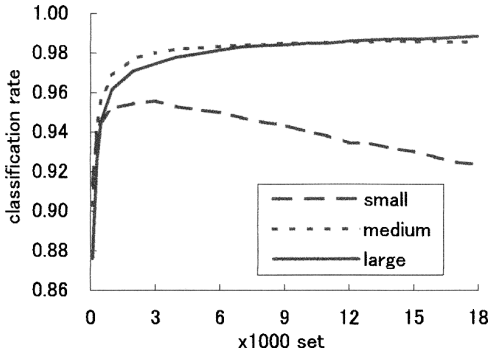


図 7 PSS の学習サンプル数と認識率の関係
Fig. 7 Relation between the number of learning samples and the classification rate.

1-NN 識別の性能を見ると，この実験で用いた特徴は大体標準的なレベルであると思われる [13]．PSS の認識率は 1-NN よりも少し劣っている．一方，PSS の識別時間は非常に高速であり，1-NN 法の 700 倍以上となっている．

4.4 BIRDS database を用いた実験

BIRDS database には，手書き数字 0〜9 が各 1 万 9,000 サンプル，合計 19 万サンプル格納されている．このうちの 18 万サンプルを学習用に，1 万サンプルを評価用に使用した．

この実験では，次元数 3 段階（small，medium，large）すべてについて性能を測定する．学習サンプル数と認識率の関係を図 7 に示す．

次元数 small では，学習サンプル数 3,000 セットで極大となり以降は認識率が下がっていく．medium では 14,000 セット以降認識率が頭打ちとなっている．この近辺が極大であると見受けられる．large では，認識率は一貫して上昇し続けており，極大には到達していない．

学習サンプルを増加させて認識率が下がるという現

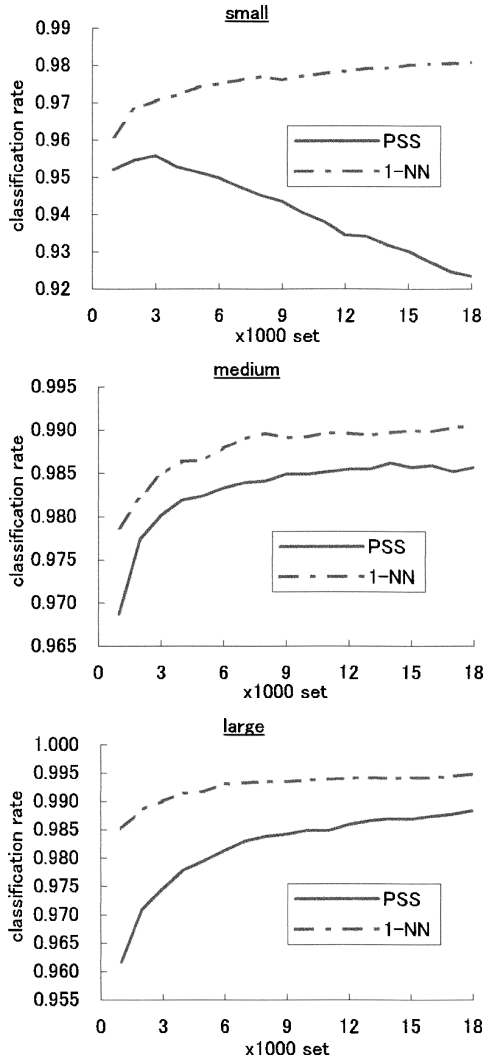


図 8　PSS と 1-NN 法の認識率
Fig. 8　Classification rates of PSS and 1-NN.

象は，**3.1** で考察したように，空間内にサンプルが密になりすぎたことによると考えられる．

次に，比較のため 1-NN 法の認識率測定結果を図 8 に示す．同じ学習サンプル数では，PSS よりも 1-NN 法の方が認識率が高い．この理由は，**3.1** で仮定した局所領域で確率密度関数が一定であるという条件に外れるためであると推測される．

学習サンプル数が同一という条件での比較では PSS は劣るが，この実験では，PSS はサンプル数が多いとき，少ないサンプル数における 1-NN の認識率を上回ることができる場合もある．NN 法はサンプル数が多

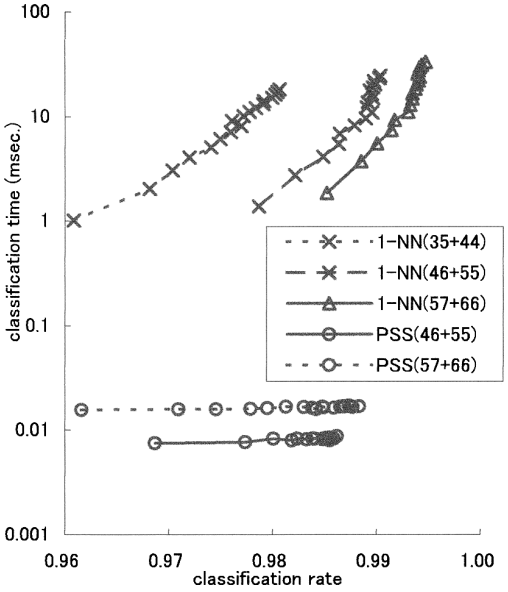


図 9　認識率と認識時間の関係
Fig. 9　Relation between the classification rate and the classification time.

いほど認識率は上昇するが，サンプル数を多くすると認識時間が線形に増加する．そこで，両者の識別時間を計測し，認識率と識別速度の関係を見ることにする．

この実験では，PSS の速度の極限を知るため，使用メモリサイズを多くし，ハッシュテーブルサイズを大きくとった．その結果サイズは約 110 MByte となった．

図 9 に結果を示す．横軸に認識率，縦軸に識別時間をとる．識別時間は 1-NN と PSS で違いが大きすぎるので対数目盛を使用している．実験より，同一の認識率では，PSS は 1-NN 法の 100 倍から 500 倍以上高速であるという結果が得られた．また，PSS の medium の識別時間は，0.0080 から 0.0087 ms，large は 0.016 から 0.017 ms と，学習サンプル数にかかわらず，ほぼ一定であった．

4.5　PSS の改良

本節では，識別率の向上のための改良方法について考察する．

先の **2.4** では，仮想的に 2^N 次元での識別関数のパラメータ学習を行っていることを示した．そこで，パラメータ b_u^k を調節して，認識率を最大化するための各種の手法が利用できる．文献 [14] では，Bagging [15] や Boosting [16], [17] などの手法を試している．

ここでは，単純アンサンブル学習の考え方を利用して，識別器サイズ，識別時間のオーダは増加せずに認

識率を向上させる手法を提示する．

たくさんの識別器を作成して，それぞれの識別器で得られた評価関数の平均を最終的な識別尺度とすることを考える．M 個のサンプルを使用して一つの識別器を作るものとするとき，今，非常にたくさんのサンプルがあるとして，M 個ずつに分ければ，多数の識別器を用意することができる．図 10 のように，非常に多数の識別器の識別結果の平均ということは，各線形識別器の重み係数の期待値で識別を行うことと等価である．

C_k に属する 1 個のサンプルで識別器を作成したとき，$b_u^k = 1$ となる確率を $p_{u,k}$ とする．C_k に属する M 個のサンプルで識別器を作成したときの $b_u^k = 1$ となる確率は，$1-(1-p_{u,k})^M$ である．したがって，極めて多数の識別器を作成して識別尺度 s_k の平均をとることは，

$$E[s_k] = \sum_{i=0}^{2^N-1} b_u^* E[b_u^k] = \sum_{i=0}^{2^N-1} b_u^* (1-(1-p_{u,k})^M)$$

となる．ただし，真の $p_{u,k}$ は分からないので，サンプルデータから推定した値 $\hat{p}_{u,k}$ を使用する．

この方法で MNIST で実験したところ，認識率は 95.94%から 96.24%に上昇した．一方，この方法による識別時間のオーダは変わらない．

しかし，この方法は識別器のデータ構造や評価関数の計算処理がやや複雑になるため識別時間は 2〜3 倍程度遅くなるという実験結果を報告している [14]．

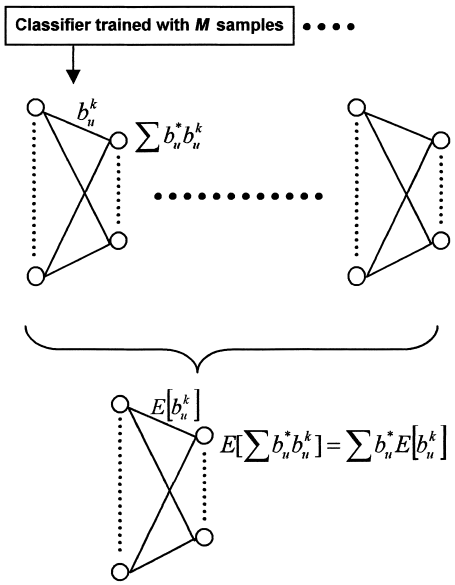


図 10　多数の識別器の統合
Fig. 10　Ensemble of many classifiers.

5. LSH との比較と考察

5.1 LSH の仕組み

ここでは，LSH によるパターン認識への適用と PSS との比較を行う．LSH は NN 探索の近似アルゴリズムである．以下に LSH の仕組みを述べる．

距離空間内の 2 点を $\boldsymbol{p}$，$\boldsymbol{q}$ とする．$d(\boldsymbol{p},\boldsymbol{q})$ を $\boldsymbol{p}$ と $\boldsymbol{q}$ の距離とする．ハッシュ関数 H が次のような性質をもつとき locality sensitive であるという．

- $d(\boldsymbol{p},\boldsymbol{q})$ が小さいとき，確率 $\Pr[\mathrm{H}(\boldsymbol{p}) = \mathrm{H}(\boldsymbol{q})]$ は 1 に極めて近い．
- $d(\boldsymbol{p},\boldsymbol{q})$ が大きくなると確率 $\Pr[\mathrm{H}(\boldsymbol{p}) = \mathrm{H}(\boldsymbol{q})]$ は急激に 0 に近づく

このような性質をもつ h 個のハッシュ関数 $\mathrm{H}_1, \mathrm{H}_2, \ldots, \mathrm{H}_h$ を用意すれば，$d(\boldsymbol{p},\boldsymbol{q})$ がある程度小さいとき，確率 $\Pr[\mathrm{H}_1(\boldsymbol{p}) = \mathrm{H}_1(\boldsymbol{q}) \text{ or } \mathrm{H}_2(\boldsymbol{p}) = \mathrm{H}_2(\boldsymbol{q}) \text{ or } \ldots \mathrm{H}_h(\boldsymbol{p}) = \mathrm{H}_h(\boldsymbol{q})]$ は 1 に十分近く，$d(\boldsymbol{p},\boldsymbol{q})$ が大きいと 0 に近いような形にすることができる．図 11 に例を示す．h はハッシュ関数の個数を表す．これによって，プロトタイプ $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m$ と任意のクエリ $\boldsymbol{q}$ が与えられたとき，$\boldsymbol{q}$ の近くに存在する点をほとんど見つけ，遠くに離れている点はあまり見つからないような検索用のデータ構造を作ることができる．そして，近くに存在する可能性の高い少数のプロトタイプとだけ正確な距離計算を行えばよいので，高速かつ高精度の NN 探索アルゴリズムが実現できる．すなわち，$\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m\}$ の中から $\mathrm{H}_1(\boldsymbol{p}_i) = \mathrm{H}_1(\boldsymbol{q})$ or

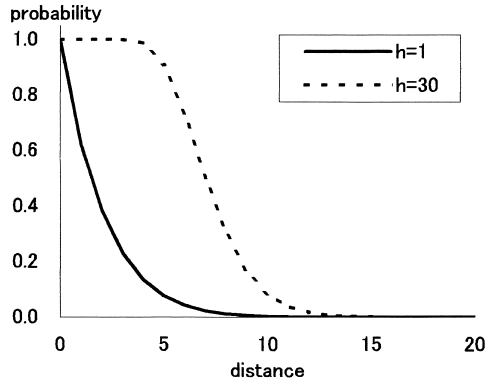


図 11　距離とハッシュ関数の一致の確率
Fig. 11　Distance and coincidence probability of locality sensitive hash functions.

$H_2(\boldsymbol{p}_i) = H_2(\boldsymbol{q}) \ldots$ or $H_h(\boldsymbol{p}_i) = H_h(\boldsymbol{q})$ であるような点 $\boldsymbol{p}_i$ を最近傍候補としてこの全体を R とする．$\boldsymbol{q}$ と R の各点について距離を計算し，一番近いものを探索結果とする．

LSH による NN 探索の計算時間は，h 回のハッシュ検索の処理と，最近傍候補との $|R|$ 回の距離計算処理の合計である．適切なハッシュ関数を選べば，プロトタイプ数 m に比べて非常に小さい距離計算回数だけで，しかも，高精度で最近傍を見つけることが可能となる．ハッシュ関数の形によって，探索誤差に対する $|R|$ の上限は求められる．

上のような性質をもつハッシュ関数の例としてはバイナリー空間におけるランダムな射影がある．

5.2 LSH によるパターン識別プログラムの実現

LSH によって返された NN 探索結果の属するクラスを識別結果とすれば，直ちにパターン識別プログラムとなる．

先の **4.** の実験では $Facto(N)$ での識別を扱っており，1-NN 法も $Facto(N)$ での距離計算を行っている．したがって，比較のために LSH を実装するには $Facto(N)$ における適切なハッシュ関数を用意する必要がある．ここでは NN 識別としての LSH と提案手法の性能を比較することが目的であるので，実験によって LSH の条件を決定している．我々はいろいろ試した結果，効果的なハッシュ関数として次のような 2 種類の例を見つけた．

ここで $\boldsymbol{f} = (f_1, f_2, \ldots, f_N) \in Facto(N)$ とする．

- ハッシュ関数（その 1）

各 $k(k = 1 \sim N-1)$ に対して $Combi(N,k)$ で $\boldsymbol{f}$ に最も近い点を $\boldsymbol{c} = (c_1, c_2, \ldots, c_N)$ とする．このとき，$H(\boldsymbol{f})$ を $\boldsymbol{c}$ の各成分を並べた 2 進数 $c_1 c_2 \ldots c_N$ とする．このハッシュ関数は PSS で用いた分散表現と同一である．ハッシュ関数の数は最大 $N-1$ 個になるが，k が 0 または N に近いとき一つのハッシュ関数でたくさんの最近傍候補を見つけてしまいがちであり，逆に k が $N/2$ に近いときほとんど見つけることができない可能性がある．したがって，効果的でない k の値はあらかじめ省いておくこともできる．

- ハッシュ関数（その 2）

$\boldsymbol{f}$ からランダムに N' 個 $(N' < N)$ の成分を取り出して $\boldsymbol{f}'$ とする．適当な k に対して $Combi(N',k)$ の中で $\boldsymbol{f}'$ に最も近い点を $\boldsymbol{c}' = (c'_1, c'_2, \ldots, c'_{N'})$ とする．このとき $H(\boldsymbol{f})$ を $\boldsymbol{c}'$ の各成分を並べた 2 進数 $c'_1 c'_2 \ldots c'_{N'}$ とする．これも，N' が小さく k が 0 または N' に近いときは一つのハッシュ関数でたくさんの点を見つけてしまいがちであり，N' が大きく k が $N'/2$ に近いときは点をほとんど見つけることができない．したがって適切な N'，k を設定する必要がある．

5.3 実験結果

前節 **5.2** に示したハッシュ関数を使用して，MNIST 及び BIRDS database で認識率と識別時間の計測を行った．

ハッシュ関数を構成するパラメータ (k, N') 及びハッシュ関数の数 h，また，学習サンプル数によって LSH の認識率と識別時間は大きく変化する．実験ではパラメータを様々に変化させた．

MNIST では，学習用サンプル 60,000 個すべてを使用して LSH の識別器を作成し，実験に用いた．結果を図 12 に示す．なお，図 12，図 13 では縦軸に対数目盛を使用している．

認識率と識別時間はトレードオフの関係にあり，わずかに認識率を低下させるだけで，識別時間を大幅に短縮できることが分かる．しかしながら，LSH は識別時間の短縮を追求していくと PSS の認識率を下回ってしまう．ほぼ同一の認識率の場合では，LSH の識別時間は PSS の約 30 倍となった．

次に，BIRDS database ではハッシュ関数のパラメータのほかに，学習サンプル数，特徴次元数をいろいろと変えて計測した．

図 13 に結果を示す．ここでもやはり，各種条件を様々に変えることにより，任意に認識率と識別速度のレベルを選択できることが確認できた．LSH は基本的な 1-NN 法の識別時間を大幅に短縮できる．それでも

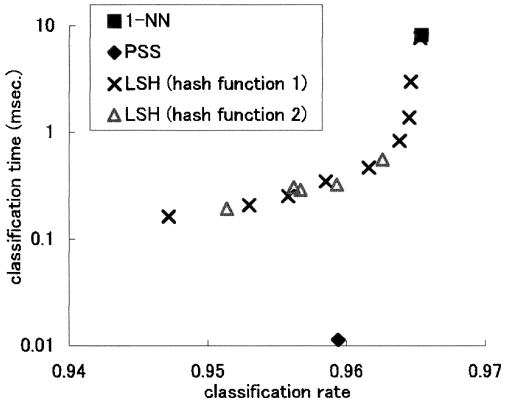


図 12 MNIST における LSH の認識率と認識時間の関係
Fig. 12 Relation between the classification rate and the classification time on MNIST.

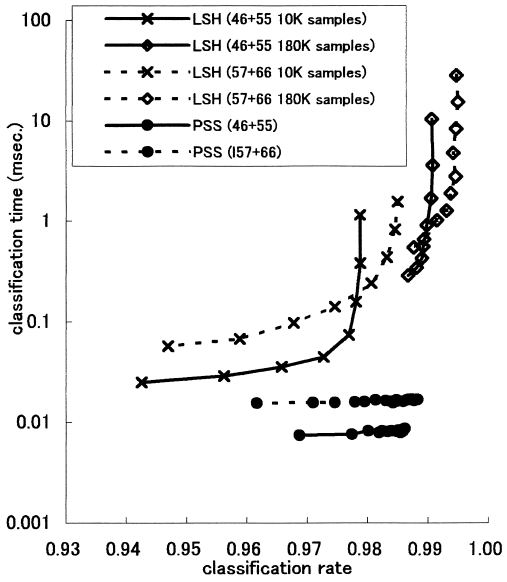


図 13　様々なパラメータでの LSH の認識性能
Fig. 13　Classification performance of LSH with various parameters.

なお，同一の認識率という条件では，LSH の最良のパラメータでも PSS より約 5〜50 倍遅いという結果になった．

5.4 考　　察

LSH では，検索して見つかった最近傍候補すべてに対して距離計算を行っている．ハッシュテーブルの検索処理は比較的高速であるのに対して，距離計算は一つのベクトルにつき N 回の乗算を行わなければならない．識別処理の時間の大部分はここで占められる．LSH は総当りの探索よりは大幅に処理時間を短縮することができる．それでも今回用いたデータベースによる実験では，学習サンプルは 1 万〜18 万個と多いので，1 パターンの認識には約 100 回から数千回程度の距離計算を行っている．したがって，およそ数千から数十万回の乗算と加算を要する．

一方，PSS ではハッシュテーブルの探索回数は $N-1$ 回であり，評価関数の計算は学習サンプル数にかかわらず最大 $N_C \times (N-1)$ 回の加算を行うにすぎない．実験で用いた値は $N_C = 10$, $N = 31, 49, 71$ であるから，1 回の識別でたかだか 300〜700 回程度の加算だけで済む．

このような理由により，LSH でいかに最適なハッシュ関数を用いたとしても PSS に匹敵するほど識別時間を短縮することは不可能と思われる．

先の **2.4** で示したとおり，PSS は線形なクラス識別関数の構造をしており，これが識別速度の点で根本的に LSH と差が出る理由である．また，PSS はなぜこのような構造で高い識別性能を示すのかという根拠，すなわち，なぜ学習サンプルとの距離計算を直接しなくて済むのかという点については **3.** で示した．したがって，PSS はベクトルを複数のベクトルでハッシュするという部分は LSH に似ているが，識別構造としては全く異なっているということができる．

なお，LSH には様々なアルゴリズムのバリエーションがある．しかしながら，本論文で扱ったデータベースで試した範囲では，これ以外で良好な認識率を示す方法を見つけることができなかった．

5.5 学習時間と識別器サイズ

LSH による識別器サイズは，ハッシュ関数の数を h とすると $O(hm)$ となる．実験ではハッシュテーブルのサイズを大きくとり，識別器のサイズは 110 MByte とした．

PSS と LSH のどちらも，プログラムによる様々な実装方法があり，識別器サイズは異なってくる．実質的には識別器サイズは両者同程度であると思われる．

学習は，PSS，LSH のどちらも学習サンプル集合を 1 回走査するだけで済む．学習用サンプルのファイルからデータを 1 件ずつ読み込みながらハッシュテーブルに追加していくので，ファイル読込みが完了した時点で識別器の作成が完了する．

PSS，LSH とも，識別器作成に要した時間は，MNIST の場合，ハードディスクからのデータ読込み処理も含むトータルの時間で約 5 秒であった．なお，1-NN 法では約 4 秒であった．

これらいずれの手法も，学習処理時間は他のパラメータ最適化学習を行う手法に比べて非常に高速である．

6. むすび

学習サンプル数に対して学習時間が線形のオーダであり，かつ，識別時間がサンプル数に対して定数オーダである学習・識別手法 PSS を提案し，理論的な考察を与えた．

更に，オフライン手書き数字認識の実験を行い，識別時間は基本的な NN 法に比べて極めて高速であることを確認した．また，高速な NN 探索手法である LSH と比較を行い，パターン識別手法としては，識別速度の点で PSS の方が十分に優位であることを確認した．

MNIST による実験では，PSS は 1-NN 法より識別率は若干低かった．一方，識別時間は PSS の方が 700 倍以上速かった．また，MNIST よりも大量のサンプルデータを用いた実験において，認識率同一の条件下で識別時間の比較をしたところ，PSS は 1-NN 法に対して 100 から 500 倍以上，LSH に対して 5 から 50 倍以上高速であるという結果を得た．

また，PSS の改良手法について提示し，識別性能が向上できることを確認した．

また，PSS の学習時間は基本的な NN 法に匹敵するほど高速であった．

提案手法は学習サンプル数に対してスケーラブルであり，逐次学習が可能なため，従来適用が難しかった大規模かつ多数クラスの識別問題ほど効果を発揮すると思われる．また，その基本原理はシンプルであるので，一般化の考察を深めることにより，様々な応用が可能になると思われる．

今後の課題としては，様々な分野の実際の問題に対して適用して有効性を検証することである．また，提案手法は狭義のパターン認識分野にとどまらず，データマイニング分野への適用も可能であると考える．この方向についても更に研究を深めていきたい．

謝辞　本研究のためにデータベースの提供をして下さった（株）バーズ情報科学研究所に感謝致します．本研究の一部は独立行政法人情報処理推進機構による未踏ソフトウェア創造事業の支援を受けている．

文　献

[1] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification, second ed., p.16, Wiley-Interscience Publication, New York, 2000.
[2] V. Vapnik, The nature of statistical learning theory, Springer-Verlag, New York, 1995.
[3] T. Kohonen, “The self-organizing map,” Proc. IEEE, vol.78, pp.1464–1480, 1990.
[4] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, Classification and Regression Trees, Chapman & Hall, New York, 1993.
[5] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, CA, 1993.
[6] T.M. Cover and P.E. Hart, “Nearest neighbor pattern classification,” IEEE Trans. Inf. Theory, vol.IT-13, no.1, pp.21–27, Jan. 1967.
[7] M. Minsky and S. Papert, Perceptrons, MIT Press, Cambridge, Mass achusetts, 1969.
[8] J.L. Bentley, “Multidimensional binary search trees used for associative searching,” Commun. ACM, vol.18, no.9, pp.509–517, 1975.
[9] P. Kanerva, Sparse Distributed Memory, MIT Press, Cambridge, Massachusetts, 1988.
[10] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” Proc. 30th ACM Symposium on Theory of Computing, pp.604–613, 1998.
[11] T. Kobayashi, “Method of and apparatus for pattern recognition and method of creating pattern recognition dictionary,” United States Patent, 5,689,584, Nov. 1997.
[12] http://yann.lecun.com/exdb/mnist/
[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol.86, no.11, pp.2278–2324, Nov. 1998.
[14] T. Kobayashi and M. Nakagawa, “Pattern recognition by distributed coding: test and analysys of the power space similarity method,” Proc. 9th IWFHR, pp.389–394, Oct. 2004.
[15] L. Breiman, “Bagging predictors,” Mach. Learn., vol.24, no.2, pp.123–140, 1996.
[16] Y. Freund and R.E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” J. Comput. Syst. Sci., vol.55, pp.119–139, 1997.
[17] 村田　昇，“Boosting の幾何学的考察，” 信学技報，PRMU2002-97, Oct. 2002.

（平成 17 年 9 月 12 日受付，18 年 3 月 6 日再受付，
7 月 25 日最終原稿受付）



小林　卓夫　（正員）

昭 63 埼玉大・理・数学卒．民間企業にて，文字認識・レイアウト解析などの研究に従事．平 18 東京農工大大学院博士課程単位取得退学．



中川　正樹　（正員）

昭 52 東大・理・物理卒．昭 54 同大大学院修士課程了．同大在学中英国 Essex 大学留学（M. Sc. in Computer Studies）．昭 54 東京農工大・工・助手．現在，教授．オンライン手書き文字認識，手書きインタフェースなどの研究に従事．理博．