修　　士　　論　　文

SVMによる将棋の詰みの予測とその応用

Prediction of mates in shogi using SVM and its application

指導教員　　**近山 隆 教授**

東京大学大学院新領域創成科学研究科
基盤情報学専攻

氏 名　　**36331 三輪 誠**

提 出 日　　**平成 17 年 1 月 31 日**

概 要

将棋において、詰みの有無を判断することは直接勝敗につながるため非常に重要な問題である。本研究では、この詰みの有無の判定問題に対する 1 つのアプローチとして、SVM を用いた将棋の詰み有無の予測の学習を行った。今回は 40000 局面のサンプルを用いてこの方法を試し、探索内で呼ばれる詰探索の探索ノード数制限を制御すること、詰探索の探索ノード数を予測する評価関数として用いること、証明数、反証数の上限を制御することを試みた。結果として、86%の正解率、平均 $10\mu sec$ 以下で分類できる非常にコストの低い分類器が得られた。また、その分類器の結果に応じて詰探索の探索ノード数制限を制御することで探索の精度を損じることなく指将棋の探索時間を 62%まで削減できた。詰探索の探索ノード数を予測する評価関数として用いた場合は、探索ノード数の削減はされるが、探索時間が大きくなるという結果になった。証明数、反証数の上限の制御については、特に両者が探索できたときの結果より、85%程度詰探索の探索時間を削減できた。

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景と目的

機械学習を用いて、強いゲームプレイヤを作るという研究 [15] は 1950 年代から行われている。このような研究において、チェッカーやバックギャモン、オセロなど比較的簡単なゲームでは人間のプロに勝つ、世界のトップレベルのコンピュータゲームプレーヤとほぼ同等の実力を持つなど、非常に良い成績を挙げている。また、チェスや将棋、囲碁 [3]、Amazon など比較的難しいゲームにおいても、ニューラルネットワークや TD 法 [2] を用いて評価関数の一部を最適化する、などゲームの一部を最適化する研究が盛んに行われており、好成績とは言えないまでも、それなりに良い結果を出している。

このような「ゲームの一部」に含まれるものとして、将棋の詰み判定が挙げられる。将棋において、詰みの有無を判断することは直接勝敗につながるため、非常に重要な要素である。そのため現在の指将棋プログラムでは、特に終盤において通常の探索の中で独自の詰みチェックアルゴリズムを呼び出し、それに多くの時間をかけることが一般的になっている。例えば、東京大学近山研究室の将棋プログラム『激指』では、全体では全探索時間の 15%、特に終盤においてはその 30%を詰みチェックに用いている。この詰みチェックアルゴリズムには、PDS、df-pn など証明数、反証数を用いた詰将棋のための探索アルゴリズムが用いられる。これらの詰探索アルゴリズムは高速であり、その有用性は現在の指将棋プログラムが詰みの発見という部分においてはプロを上回っているといわれていることからも明らかである。

このような詰探索アルゴリズムは高速であるとは言え、最良ではなく、利用の際にはまだまだ多くの問題が残されている。

まず、詰み、不詰の最小の証明木そのものに比べると詰探索アルゴリズムは数十倍から数百倍の余分なノードを探索していることがわかっており [12][25]、その探索には改善の余地がある。

また、この詰探索の特徴として、詰みがないことの発見に関しては、その証明木が OR 木になるため、詰みの発見に比べてその探索に多くの探索ノード数を必要とすることが挙げられる。このために、探索量を多く割り当てることが必要となるが、それでも結果がわからないときには不明という結果しか得られず、多くの探索時間をほとんど意味のない探索に費やしてしまうことにもなりかねない。このような点は詰探索アルゴリズムを利用する際、問題となる部分であり、改善の余地がある。

最後に、指将棋プログラムの探索木の全ノードにおいて詰探索アルゴリズムを呼び出すには、詰探索アルゴリズムは十分に高速であるとは言えず、詰み、不詰の証明木が非常に大きくなるノードもあるため、その探索ノード数を制限する必要がある。この制限の制御については人手での静的評価や探索木の深さのみで決定する [18][27] という方法が一般的であり、ある決まった詰みに関する指

標に基づいた方法は今のところ提案されていない。

このようなことから、本研究では、詰みの有無の判定問題に対する 1 つのアプローチとして、人手でのチューニングの代替となる、Support Vector Machine (SVM) を用いた将棋の詰み有無の予測の学習手法を提案する。

また、この学習の結果に基づいた詰みの確率という、これまでの詰みの指標とは異なった新しい指標を作成する。

1.2 本研究の位置づけ

我々の研究では、ゲームにおける評価関数を半自動的に学習するための枠組みの作成を最終的な目的としている。この枠組みには、ゲームに関する特徴量を半自動的に抽出、選別することとその重み付けを自動的に行うことの 2 つが主な要素として含まれている。

本論文ではこの重み付けを自動的に行う手法の一例として、将棋の詰みにおけるその有無の予測を学習する方法について提案し、その手法により得られた評価関数の用途、効果について実験による検証を行う。

1.3 本論文の構成

以降、本論文の構成は次のようになっている。

第 2 章 関連研究

機械学習の将棋への応用、詰探索、SVM といった関連研究について述べる。

第 3 章 Support Vector Machine による詰みの予測

本研究の基本となる詰み予測を行う分類器の構築とその評価について述べる。

第 4 章 詰み予測の応用

詰み予測の応用として、詰探索 PDS の探索ノード数制限の制御、PDS の評価関数への適用、PDS の証明数、反証数の上限の制御についての説明と、その評価について述べる。

第 5 章 結論

本研究の結論と今後の課題について述べる。

第2章　関連研究

本章では関連研究として、2.1 において機械学習の将棋への適用事例について紹介する。次に 2.2 において詰探索アルゴリズムについて説明し、最後に 2.3 において本研究で用いた分類器である Support Vector Machine について説明する。

2.1　機械学習の将棋への適用

機械学習を用いてコンピュータ将棋プレイヤを強くするという研究は広く行われており、パターン認識 [7] [9] を用いたもの、ニューラルネットワークを用いたもの [26][8] TD 法 [19] を用いたもの、統計情報 [23] [24][17] を元にしたもの、などがある。ここでは、今回の研究で用いた SVM にその用途の近いニューラルネットワークを用いたものを 2 つ紹介する。

川田ら [26] は単層、3 層パーセプトロンを用いて、局面の進行度の評価を行った。入力には 35 要素の特徴ベクトルを用い、教師信号としては初期局面を 0.1、終了局面を 0.9 とした線形の信号を用いている。データには 200 個の平均手数 100 手くらいの比較的手数の短い棋譜を用い、学習データを 100 個にした場合、150 個にした場合の 2 つで評価を行っている。結果は図 2.1 のようになっている。結果を見ると 3 層パーセトロン、150 個を学習データとした最もよい場合でも、その誤差は 0.6 と非常に大きくなっていることが分かる。

Grimbergen[8] は単層パーセプトロンを用いて、王の危険度の評価を行った。入力には 161 要素の特徴ベクトルを用い、500 個の中盤の局面を正例、500 個の詰将棋を負例として、そのデータを学習データとテストデータにわけ、学習率 0.01 と 0.001 それぞれについて、学習回数を 5000 回から 50000 回まで 5000 ずつ変化させて評価を行っている。結果は図 2.2 のようになっており、これは学習回数を 50000 回としたときの結果であるが、学習データが多いときは 80%以上の正解率を示していることがわかる。

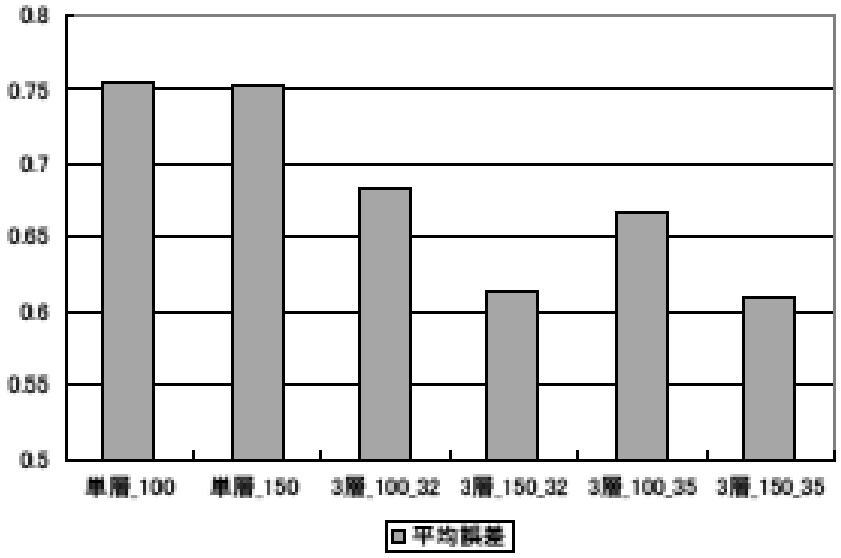

図 2.1: 局面進行度の評価

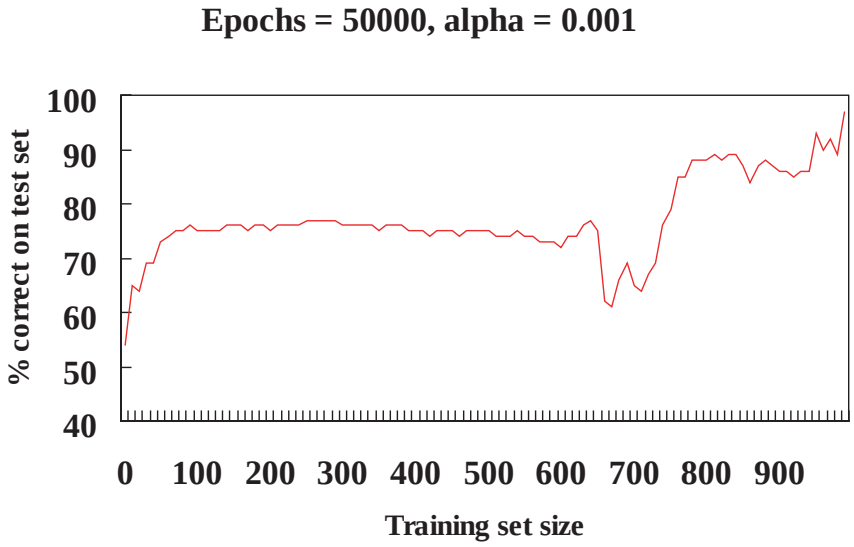

図 2.2: Learning Curves for the perceptron. (Grimbergen, 2002)

2.2 詰探索アルゴリズム

本節では、現在多くの指将棋プログラムで用いられている詰探索アルゴリズムについて説明する。まず、2.2.1 において探索する対象である詰みとは何かについて説明し、2.2.2 において詰探索アルゴリズムとその現在最もよく用いられているものの 1 つである PDS について説明したのち、2.2.3 において実際の指将棋のゲーム木探索の内部での詰探索アルゴリズムの利用について説明する。

2.2.1 詰み

将棋において、詰みは狭義では次のように定義することができる。

Definition 1 狭義の詰み
どんな合法の手を指しても、次の相手の手において玉が相手に取られる局面。

図 2.3 に狭義の詰みの一例を示す。この局面では、玉がどこに動いても金、もしくは歩に取られてしまうため詰みである。

また、広義において詰みは次のように定義することができる。

Definition 2 広義の詰み
連続王手で正しい手順を指す (「詰ます」) ことで、狭義の詰みに持ち込める局面。

この定義では、詰将棋全般は詰みであるということができる。図 2.4 に広義の詰みの例を示した。

2.2.2 PDS

詰探索アルゴリズムとは、探索を行うことで対象の局面が広義の詰みであるかどうかを判断するアルゴリズムである。詰探索アルゴリズムはその詰み・不詰の結果と、もし対象の局面が詰みであったならば、狭義の詰みに持ち込むまでの手順が 1 つ以上返す。実際には現実時間ですべての局面の詰み・不詰問題を解くことは現在の環境では不可能であるので、探索の上限を決め、それを超えた場合は不明という結果を返すことを行うのが一般的である。

このような詰探索アルゴリズムは、主に詰将棋を対象として、1970 年代からゲーム木の問題の 1 つとして alpha-beta 法を元に研究がなされてきたが、1990 年代始め AND/OR 木探索の問題として考えられるようになってから急速に発展してきた。特に共謀数 [11] を基にした証明数・反証数を用いた探索手法は、1997 年に図 2.4 に示した 1,525 手詰のミクロコスモスを解くことに成功するなど、非常に良い結果 [16] を示している。このような詰探索アルゴリズムには PDS(Proof-number and Disproof-number search) [12] や df-pn(depth first proof number search) [13] があげられる。

ここでは、本研究で用いた詰探索アルゴリズム PDS について紹介する。

PDS(Proof-number and Disproof-number search) [12] とは、長井によって 1998 年に提案された深さ優先の AND/OR 木探索手法である。PDS は、できるだけ少ないノードの展開でそのルートノー

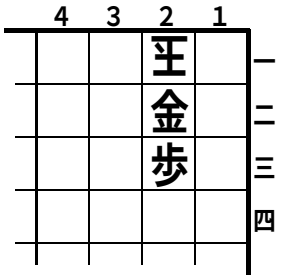

図 2.3: 狭義の詰み

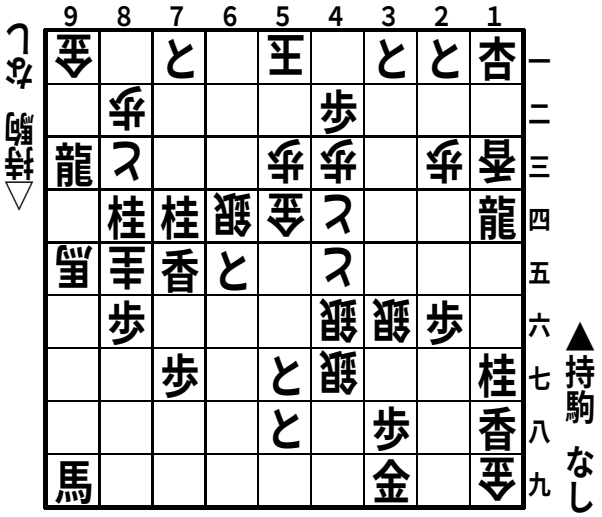

図 2.4: 広義の詰み (ミクロコスモス)

```
// ルートノード r において
// Iterative deepening
procedure NegaPDS(r) {
  r.φ=1; r.δ=1;
  while (r.φ≠0 && r.δ≠0) {
    MID(r);
    // 証明数、反証数の小さいほうの
    // 上限を増やして探索
    if (r.φ≤r.δ) r.φ++;
    else r.δ++;
  }
}

// Multiple Iterative Deepening
procedure MID(n) {
  // Transposition table を見て、
  // 必要なければ探索打ち切り
  LookUpTT(n, &φ, &δ);
  if (φ = 0 || δ = 0 ||
    (φ≥n.φ && δ≥n.δ)) {
    n.φ = φ; n.δ = δ;
    return;
  }
  // 証明 (反証) されたときは探索終了
  if (n is a terminal node) {
    if ((n.value= true &&
      n is an AND node) ||
      (n.value= false &&
      n is an OR node)) {
      n.φ = inf; n.δ = 0;
    } else {n.φ = 0; n.δ = inf;}
    PutInTT(n, n.φ, n.δ);
    return;
  }
  // 証明 (反証) されなかったときは
  // 全ての合法手を作成
  gen_all_moves();
  // サイクルの発見のため
  PutInTT(n, n.φ, n.δ);
  while (1) {
    // threshold で打ち切り
    if( ΦSum(n)=0 || ΔMin(n)=0 ||
      (n.φ≤ΦSum(n) && n.δ≤ΔMin(n))) {
      n.φ = ΦSum(n);  n.δ = ΔMin(n);
      PutInTT(n, n.φ, n.δ);
      return;
    }
    φ = max(φ, Δ(n));
    n_child = SelectChild(n, φ);
    LookUpTT(n_child, &φ_child, &δ_child)
    if (n.φ>ΦSum(n) &&
      (φ_child≤δ_child || n.φ≥ΔMin(n))) {
      n_child.φ = φ_child + 1; n_child.δ = δ_child;
    } else {
      n_child.δ = δ_child + 1; n_child.φ = φ_child;
    }
  }
  MID(n_child);
}

Procedure SelectChild(n, φ) {
  for ( each child n_child ){
    δ_child = n_child.δ;
    if ( δ_child≠0 )δ_child = max(δ_child, φ);
    return the child with minumum n_child.φ
            among children with minumum δ_child
  }
}
```

図 2.5: PDS アルゴリズム

ドが true, false(それぞれ将棋では詰み, 不詰に対応) であることを証明しようとする。そのためにPDS は、ルートノードが true, false であることを証明するために必要な展開ノード数が、少ないほうを優先して探索を行っていく。

PDS は次のような特徴を持ったアルゴリズムである。

- 証明数、反証数の両方を用いる。
- 探索結果を覚えておく Transposition Table を用いて、ルートノードから多重反復深化 (multiple iterative deepening) を行うことで、比較的少ないメモリ量で (証明数、反証数が少ないということを最良とした) 最良優先探索と同等の効率的な探索ができる。

ここで、証明数、反証数とは次のように定義される数である。証明数 (proof-number, pn) とは、あるノード n が true(将棋では詰み) であることを証明するには、そのノードから派生した木の最低あといくつの葉ノードが true であることが証明される必要があるか、を示す値であり、次のように定義される。

Definition 3 証明数

1. n が葉ノードのとき
   (a) 評価値が true のとき $n.pn = 0$
   (b) 評価値が false のとき $n.pn = +\infty$
   (c) 評価値が不明のとき $n.pn = 1$
2. n が内部ノードのとき
   (a) OR ノードのとき
   $$n.pn = \min_{n_c \in children\ of\ n} n_c.pn$$
   (b) AND ノードのとき
   $$n.pn = \sum_{n_c \in children\ of\ n} n_c.pn$$

同様に、反証数 (disproof-number, dn) とは、あるノード n が false(将棋では不詰) であることを証明するには、そのノードから派生した木の最低あといくつの葉ノードが false であることが証明される必要があるか、を示す値であり、次のように定義される。

Definition 4 反証数

1. n が葉ノードのとき
   (a) 評価値が true のとき $n.dn = +infty$
   (b) 評価値が false のとき $n.dn = 0$
   (c) 評価値が不明のとき $n.dn = 1$
2. n が内部ノードのとき
   (a) OR ノードのとき
   $$n.dn = \sum_{n_c \in children\ of\ n} n_c.dn$$

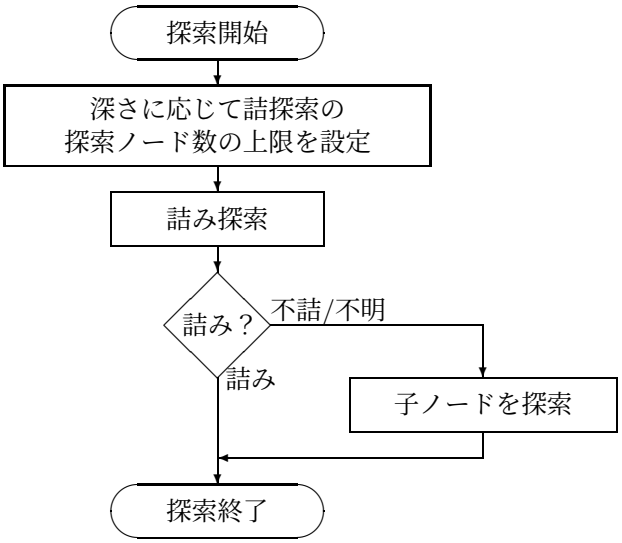

図 2.6: 指将棋プログラムの流れ

(b) AND ノードのとき

$$n.dn = \min_{n_c \in children\ of\ n} n_c.dn$$

PDS アルゴリズムを図 2.5 に示した。PDS は上記のように定義される証明数、反証数を用いて次のように探索する。まず、証明数、反証数の閾値をそれぞれ 1 に設定する。そして、ルートノードから証明数もしくは反証数の小さいノードから、ノードの閾値をルートの閾値を越えないように徐々に増やしながら順に展開する。探索中、前に探索して探索が不要なノード、証明数、反証数が閾値を超えたノードについてはそこで展開を打ち切る。このように展開して、ルートノードの証明数、反証数が閾値を超えた場合は、証明数、反証数いずれかの閾値を増やしてルートノードから再び探索を行う。このような探索を繰り返し、ルートノードの証明数もしくは反証数が 0 になった場合、それぞれルートノードが true, false であるとわかるので、探索終了となる。

2.2.3 指将棋プログラム内での詰探索アルゴリズムの利用

詰探索は、通常のゲーム木探索に比べて、非常に高速である。そのため、指将棋プログラム内では、一般にいわゆる「必至」のような連続王手でない詰みや、こちらが何もしなければ詰まされてしまう「詰めろ」をみつけるために探索中のノードほとんどすべてにおいて詰探索を行う。ノード内での探索の様子を図 2.6 に示す。この詰探索は浅いところでは多く、深いところでは少なくノード数を制限して探索を行い、深すぎるところでは時間の都合上呼ばれないことが多い。

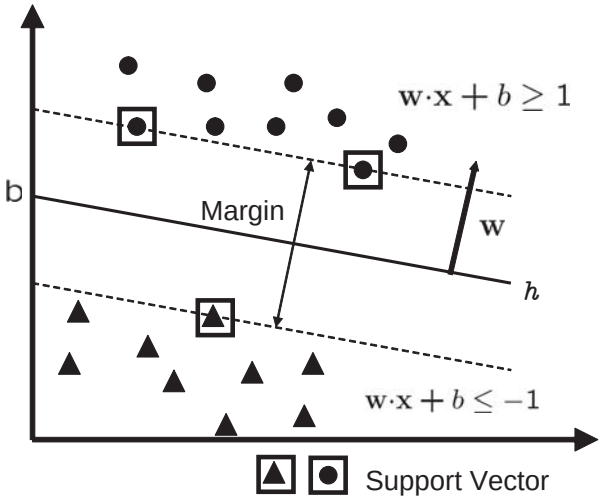

図 2.7: Support Vector Machine

2.3 Support Vector Machine

Support Vector Machine(SVM)[6] は、1992 年に Vaptik らによって提案された 2 クラスの分類器の 1 つである。

SVM は、超平面とそれに最も近い例との距離 (マージン) を最大化するように、学習データを線形分離する (分離) 超平面を求める。その特徴として主に次の 3 つが挙げられる。

- 少ないサポートベクタで分割境界の超平面を張ることで、計算量を少なくできる。
- 非線形領域へのカーネル関数を用いたマッピング (カーネルトリック) により、非線形の分割問題を計算量をあまり増やさずに分割可能な問題に変換できる。
- 出力をシグモイド関数に当てはめることで、確率としても利用可能である。[14]

この中の、カーネルトリックについては本研究では用いなかったのでここでは説明しない。本節では 2.3.1 において線形 SVM について説明したのち、2.3.2 においてその識別誤りを吸収する手法として提案されているソフトマージンアルゴリズムについて説明する。

2.3.1 線形 SVM

SVM では図 2.7 のように特徴空間上にある 2 クラスの集合に対して、それを分離する超平面 h を動かして、その超平面とそれに最も近い例との距離 (マージン) が最大になるように分類器を構築する。そのマージンが最大である超平面を分離超平面と呼ぶ。この分離超平面が未知のデータを分類する基準であり、分類器そのものとも言えるものであるので、ここで分離超平面を求める方法について説明する。

図 2.7 のように、$\mathbf{w}$ を超平面の法線ベクトルとし、$\mathbf{x}_i$ が正例ならば+1、負例なら-1 となるものを y_i とする。

このとき、マージンの長さは $2/||\mathbf{w}||$ となるので、マージンを最大化するということは $||\mathbf{w}||$ を最小化すればよい。そのため、問題が線形分離可能であれば、以下のように表現できる。

$$\begin{aligned} &\text{Minimize } ||\mathbf{w}|| \\ &\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \end{aligned} \tag{2.1}$$

この問題はラグランジュ未定乗数を導入することにより以下の双対問題に変形できる。

$$\begin{aligned} &\text{Minimize } -\sum_{i=1}^{n}\alpha_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j \\ &\text{subject to } \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } \forall_i \alpha_i \geq 0 \end{aligned} \tag{2.2}$$

このとき Karush-Kuhn-Tucker 条件

$$\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1) = 0 \tag{2.3}$$

が要求されるため、超平面から一番近いデータの α 以外は全て 0 になる。これらの超平面から一番近いデータは、超平面を構成する要素であるため、サポートベクタ (Support Vector, SV) と呼ばれる。

この式 (2.2) の最適解 α^* を用いて

$$\mathbf{w}^* = \sum_{\mathbf{x}_i\in SV}\alpha_i^* y_i \mathbf{x}_i \tag{2.4}$$

$$b = y_k - \mathbf{w}^*\cdot\mathbf{x}_k,\ \mathbf{x}_k\in SV \tag{2.5}$$

と求めることができる。これにより、分類器の判別関数 (分離超平面の関数) は

$$f(\mathbf{x}) = sgn(\mathbf{w}^*\cdot\mathbf{x} + b) \tag{2.6}$$

$$= sgn\left(\sum_{\mathbf{x}_i\in SV}\alpha_i^* y_i(\mathbf{x}_i\cdot\mathbf{x}) + b\right) \tag{2.7}$$

と求めることができる。このように分類器の判別関数は SV のみを用いて計算すればよく、また一般に SV は入力データの数に対して非常に少なくなることが多いため計算量を少なくすることができる。また、(線形においては) 事前に $\mathbf{w}^*$ を計算しておくことにより、一度のベクトルの掛け算のみで分類ができ、非常に少ない計算量で分類が可能となる。

ところで、これまで、マージンを最大化すればよいということを前提に話を進めてきた。しかし、なぜこのような方法をとっているのか、ということについては何も触れていなかったので、ここで簡単にだが説明しておく。分類器の判別関数 $f(\mathbf{x})$ についてその誤りの期待値をリスクといい、クラス

$y \in \{+1, -1\}$ の事前確率を $P(y)$、クラス y の特徴空間 $\mathbf{x}$ 上での確率密度を $p(\mathbf{x}|y)$ とすると次のように表すことができる。

$$P_{true\ error} = \sum_{y \in \{+1,-1\}} \int \frac{1}{2} |f(\mathbf{x}) - y| p(\mathbf{x}|y) P(y) d\mathbf{x} \tag{2.8}$$

各クラスの確率に関する情報を得ることができれば、この式 (2.8) を用いて計算すればよいが、一般にはそのような情報を得ることは難しい。そのため、そのような場合に対処するための方法として構造的リスク最小化 (Structual Risk Minimization) の原理が提案されている。学習データから得られる誤り率を経験リスクといい、学習データ数 l について、次のように表すことができる。

$$P_{train\ error} = \frac{1}{2l} \sum_{l}^{i=1} |f(\mathbf{x}) - y_i| \tag{2.9}$$

これらの $P_{true\ error}$、$P_{train\ error}$ について、任意の $\lambda (0 < \lambda < 1)$ に対して 1-λ の確率で

$$P_{true\ error} \leq P_{train\ error} + \sqrt{\frac{d(log(2l/d) + 1) - log(\lambda/4)}{l}} \tag{2.10}$$

となることが知られている。ここで d を Vapnik Chervonekis(VC) dimension といい、対象とする仮説空間の記述能力・複雑さを表す尺度である。

Definition 5 VC dimension
仮説空間の VC dimension とは、d 個の異なる事例に対してそれぞれ正例、負例の任意のラベル付けが可能な最大の d のことである。

この式 (2.10) の右辺第二項は、学習データ数が大きくなり VC dimension が小さくなるほど小さくなる。このことは VC dimension が大きくなれば、学習データに当てはまってしまう事例が増えることからも直感的に理解できる。また、この第二項が小さくなるほど経験リスクの精度は高くなる。

ここで構造的リスク最小化の原理とは、VC dimension(d_n) が単調増加するような関数集合の列、

$$S_1 \subset S_2 \subset \cdots S_n \subset \cdots$$

について式 (2.10) の右辺を最小化する d_n と $f \in S_n$ を求めようとするものであり、この式 (2.10) の右辺を最小にする n を選択した上で、経験リスクを最小にする決定関数を選ぶことである。

この原理にのっとれば、リスクそのものを最小化しているものではないため最良ではないものの、その最大値を最小化することができるため、汎化能力の高い分類器が構築できることが期待される。

SVM では d は、すべての入力ベクトルを含む最小の球の半径を R、特徴空間の次元を n とすると、

$$d \leq min(R^2 ||\mathbf{w}||^2,\ n) + 1 \tag{2.11}$$

と制限されることが知られている。この式 (2.11) より $||\mathbf{w}||$ を小さくすること (マージン最大化) は d を小さくするということである。特に線形分離可能な学習データが与えられた場合、$P_{train\ error}$ は 0 となるので、SVM は構造的リスク最小化の原理に基づいた判別関数の選択を行っているということがわかる。

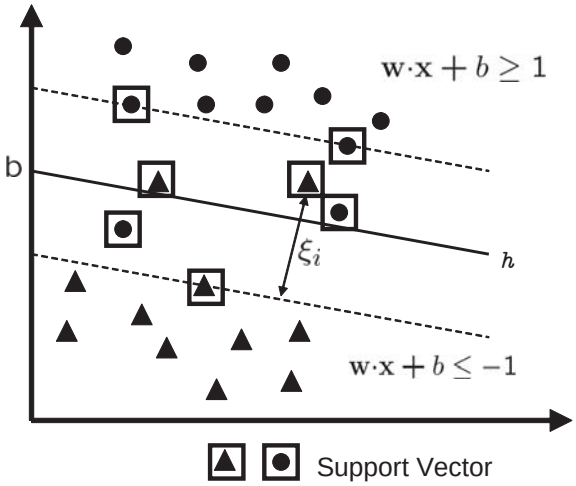

図 2.8: ソフトマージンアルゴリズム (C-SVC)

2.3.2 ソフトマージンアルゴリズム

一般に多くの問題は線形分離不可能であることが多い。このような場合に用いられる手法の一つとしてソフトマージンアルゴリズムが挙げられる。ソフトマージンアルゴリズムとは多少の識別誤りを許すように制約を緩める手法である。その緩めるパラメータの選択により、C-SVC, ν-SVC[4] などがあげられるが、ここでは C-SVC について取り上げる。

C-SVC では、図 2.8 のようにその緩めのパラメータに C, ξ を用いる。これらを用いることで、式 (2.1) の問題は次式のようになる。

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i \tag{2.12}$$

subject to

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ C > 0 \tag{2.13}$$

ここで、C は識別誤りに対するペナルティ、ξ は緩めの度合いである。また、C は経験的に決められる量であり、効率的な C を見つける方法はヒューリスティクスを用いたものが提案されてはいるが、汎用に適用可能な手法であるとは言えず、現在のところ確立されていない。この式 (2.12) は式 (2.1) と同様に解くことができる。解 (分類器) は識別誤りを SV に加えたものを SV*とすると式 (2.14) となる。

$$f(\mathbf{x}) = sgn\left(\sum_{\mathbf{x}_i \in SV*} \alpha_i^* y_i(\mathbf{x}_i \cdot \mathbf{x}) + b\right)$$

$$\text{subject to } 0 \leq \alpha^* \leq C \tag{2.14}$$

式 (2.14) は、式 (2.6) と同様に (線形領域では)$\sum_{\mathbf{x}_i \in SV*} \alpha_i^* y_i \mathbf{x}_i$ が事前計算可能であり、ソフトマージンアルゴリズムは分類に関してはその計算量を増やすことなく、多少の識別誤りを吸収することのできる手法であるということができる。

ソフトマージンアルゴリズムのペナルティは 2.3.1 に述べた経験的リスクを小さくするものであり、構造的リスク最小化の原理の指針にのっとった手法であることが分かる。とはいうものの、このソフトマージンに関しては、「線形分離可能な問題に対して」という前提が崩れているため、2.3.1 に示した理論が成り立つわけではない。それでもなお、この分類器が有用であることは、実用上で多くの分野でその有効性が示されており、また、現在提案されている分類器の中では汎化能力の高い分類器であるということが (実験的にだが) 示されていることからも明らかである。

第3章　SVMによる詰みの予測

本章では 3.1 で、本提案手法の裏づけと SVM を用いてどのように詰みの予測を行うかについて説明する。その後、3.2 において詰みの定義、特徴量の選択、学習データと実験環境について述べた後、3.3 において詰み予測を行う分類器そのものの能力についての実験結果を述べる。

3.1　提案手法

本研究では線形の SVM による詰みの予測を行う。本節では、3.1.1 で提案手法の実験的な裏づけについて述べたのち、3.1.2 でその手法の説明を行う。

3.1.1　背景

ここでは、実験的な裏づけを含め、線形の SVM を用いて将棋の詰みの有無の予測を行うことについて、次の 3 つについて説明する。

- なぜ分類器を用いるのか。(3.1.1.1)
- なぜ SVM を用いるのか。(3.1.1.2)
- なぜ非線形[1]の SVM ではなく、線形の SVM を用いるのか。(3.1.1.3)

3.1.1.1　分類器の利用

ここではなぜ分類器を用いるのかについて説明する。
分類器の利用には次の 2 つの理由がある。

- 詰み、不詰が定義しやすい明快な基準であること。
- 基準は明確ではないが、実際に詰みそう、詰まなそうな局面が存在すること。

1 つ目の理由については、詰み、不詰以外の基準としては、詰みの最短手順の長さ、詰みの証明木の最小リーフ数が挙げられる。まず、詰みの最短手順の長さについてはその詰みチェックの容易さと対応していないため決定的な基準とはいえないことは明らかである。また、詰みの証明木の最小リーフ数については、現在の詰探索の基準であり有用であることは考えられる [25] が、

[1] ここで言っている非線形とは、カーネルトリックを使ったという意味である。

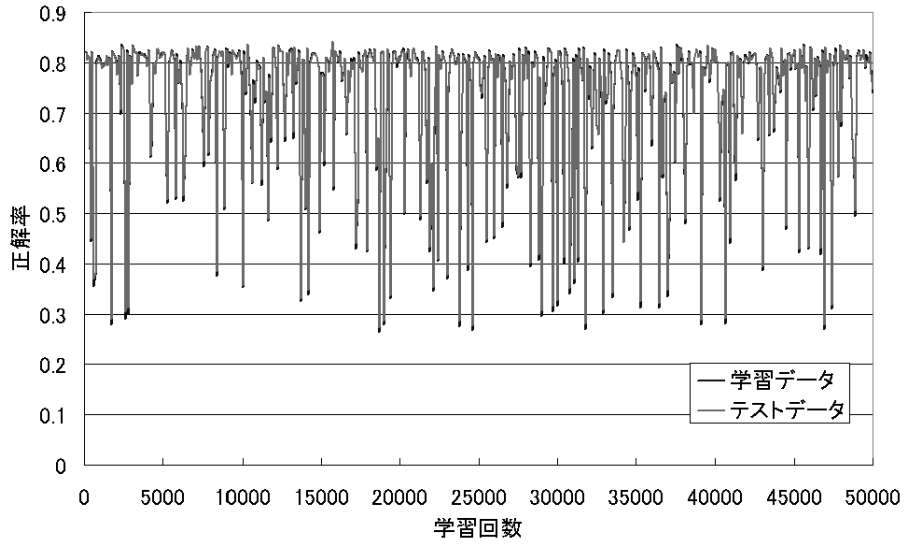

図 3.1: パーセプトロンでの学習 (α=0.01)

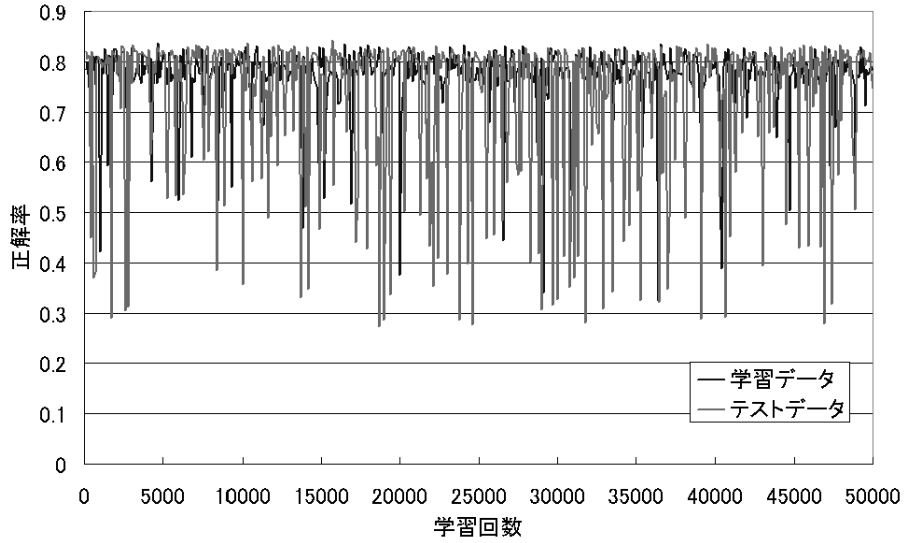

図 3.2: パーセプトロンでの学習 (α=0.001)

- 学習に十分必要な局面を用意することが難しいこと。
- 証明木に含まれるループのために、証明数、反証数の定義以外に証明木の最小リーフ数の定義が必要であること。
- 特に大きな証明木ではその最小リーフ数の計算に時間がかかること。

などの理由から今回は用いることをしなかった。

2 つ目の理由については、その基準は明らかではないが、人間はその局面を見て瞬時に詰み、不詰を判断することが言われており、詰み、不詰を分類問題として扱うことができるのではないかと考えたということである。

このように、詰みは分類問題として扱えると考えられ、また、その基準が明快なものであるため、分類器を用いた。

3.1.1.2 SVM の利用

ここではなぜ SVM を用いるかについて説明する。

これには 2.3 に示した SVM の持つ次の 3 つの特徴が理由として挙げられる。

- 一度のベクトルの内積のみで分類が可能であり非常に高速に分類できること。
- 汎化能力に優れた分類器であること。
- 出力をシグモイド関数に当てはめることにより確率として扱うことができること。

最初の 2 つの特徴は、時間制限があり、(将棋で起こりうる局面数全体に比べて) 学習に使えるサンプル数が少ないといえる将棋のような環境においては必要不可欠であるということができる。高速に分類ができるだけであれば、例えばニューラルネットワークが挙げられるが、2.1 に示した Grimbergen の実験と同様に学習率 α、学習回数それぞれについて変化させて、3.2.3 のデータを 30000 の学習データと 10000 のテストデータに分けて、単層パーセプトロンで学習させたところ、図 3.1, 3.2 のような結果が得られた。この結果から、単層パーセプトロンを用いた場合では最もよい場合でも SVM よりも 3%程度低い正解率となることが分かった。パラメータを柔軟に変化させることで、運良く SVM と同等の分類器が構築できる可能性がないわけではない。しかし、この図を見れば分かるように学習に収束の傾向は見られない上に、最良の場合でも SVM で構築できた分類器より悪いという知見が得られていることからも、汎化能力の低い分類器ではその収束のためのパラメータの調整が困難であり、この結果は汎化能力に優れたものが重要であることを示していると考えられる。この問題は線形分離不可能な問題であるため、2.3.2 で説明したとおり理論的な裏づけはないが、この結果はまた SVM の汎化能力の裏づけを強固にしているとも言える。

また、出力はシグモイド関数に当てはめることで詰みの確率としてみることができるという SVM の特徴は、本研究には必要な特徴であると考えられる。なぜならば、詰み問題が線形分離可能な問題であるかということは今のところ分かっていないが、SVM で 100%の詰み判定を行うことは非常に困難であろうことは明らかであるためである。

また、詰みの確率として扱うことにより、これまでの詰み問題に存在している判断基準 (例えば詰みの最短手順の長さ、詰みの証明木の最小リーフ数など) 以外の基準を構築することができることが期待される。また、それらの基準と今回得られる基準にどのような相関があるのかということを考察することは意味がある。特に詰みの証明木の最小リーフ数は現在の詰探索アルゴリズムの証明数と直に結びつく概念であり、この評価値を別の基準から考察することは価値のあるものであろう。

このように SVM は、高速で扱いやすく、有意義な結果を得られる分類器であるため、SVM を用いることにした。

3.1.1.3 線形 SVM の利用

ここでは、なぜ非線形ではなく、線形の SVM を用いるかについて説明する。

これには次の 2 つの理由が挙げられる。

- 非線形は線形に比べて分類に時間がかかってしまうため。
- 本研究で対象としたデータについては非線形と線形について分類能力にあまり差が見られなかったため。

1 つめの理由について厳密に説明を行うには非線形の SVM について説明しなければならないので、ここでは簡単にだけ説明することにする。非線形での分類器についても式 (2.6) に似た形になるのだが、この w^* が事前に計算できないため、非線形では Support Vector の数を nSV とするとおよそ nSV 倍の時間がかかることになる。(単純には、この nSV は超平面を構成するものであるため、(次元数+1) 以上である。) また、線形に比べて非線形のほうが分類する分離超平面の次元が大きくなるため、この nSV は一般に大きくなることが多い。この nSV を減らす研究も多くなされている [10][20] が、その分類にかかる時間が線形に比べて大きいことは否めない。

次に 2 つ目の理由についてであるが、これは事前実験として非線形の SVM についてパラメータ、カーネルを変えて、分類器を構築した結果である。この実験では、最良の場合でも 3.3 に比べて 1, 2%程度しか判別能力は向上しなかった。

将棋は時間制限のある問題であるため、用いる分類器はできるだけ高速であるほうがよいことは明らかである。非線形の SVM を用いて判別能力が極端に向上するようなことがあれば、その分類器を用いた方が 4 に示したような他の手法でその分類の間違いを補う方法より有用であることがないとはいえない。しかし、実際にはそのような向上はなく、その若干の変化を nSV 倍時間のかかる非線形の SVM を用いて補ったところで、その間違いを補う方法を取らなければならないのは同じである。

このように効果に比べて時間の効果が大きいため、今回は非線形の SVM は扱わないことにした。

3.1.2 SVM による詰みの予測

本研究では SVM による詰みの予測を行い、その予測を様々な局面に応用することを行う。応用については 4.1 で説明する。

詰みの予測を行うにあたって、その学習方法は次のようになる。

1. 詰みの基準に基づき、学習する局面に詰み・不詰のラベルをつける。
2. 学習する局面から特徴量を取り出し、ラベルを元に正例、負例として SVM に入力する。

ここで、どのような局面を詰みと判断するか、特徴量はどのような量を取り出すのかという問題があるが、これは 3.2 で説明する。

また、その予測については、予測する局面から特徴量を取り出し、SVM に入力することで予測をする、という方法になる。

3.2 実験方法

本節では、評価に用いた設定について述べる。3.2.1 で詰みの定義、3.2.2 で特徴量の選択、3.2.3 で評価に用いたデータ、3.2.4 で実験環境についてそれぞれ説明する。

3.2.1 詰みの定義

2.2.1 に示したとおり、詰みを定義することはできるが、狭義の詰みは王が動けるか否かの判定を直接将棋盤を見て判断するだけでよいため、詰み判断に用いる定義としては狭すぎる。また、その一方で広義の詰みは詰み判断に用いる定義としては広すぎる。これは、図 2.4 のような難解な問題は詰みと判断できてもコンピュータ将棋の一般的な制限時間である 30 秒以内に探索するのは (少なくとも現在の環境では) 不可能なためである。

このようにどちらの定義も実際のコンピュータ将棋の詰み判断に用いるには適切でない定義であるため、本研究では次のように詰みを定義し、この定義に合わないものは不詰とした。

Definition 6 詰み
(近山・田浦研究室の将棋プログラム『激指』実装の)PDS で 5,000,000 ノード探索するうちに広義の詰みであると判定された局面。

ここで、5,000,000 ノードとは、現在の実験環境下で約 30 秒強で探索が終わる値である。より正確な詰む詰まないの情報を得るためにこれ以上のノード数を制限に用いることも考えられるが、

- 詰む詰まないを正確に判断することは現実的に不可能であること
- コンピュータ将棋では制限時間が 30 秒に制限されており、詰探索に費やすことのできる時間は最大でも 30 秒であるため、ゲーム中には PDS で探索できず不明と判断され SVM の判断を確かめることができないため不詰と考えても差し支えないこと

の 2 つの理由からこのような値を用いた。

3.2.2 特徴量の選択

SVM に入れる特徴量としては、将棋盤そのものを入力ベクトルとして入れて、カーネルトリック [6] などにより高次元に射影して分類することも考えられるが、単純な射影では分類がまったくできない、3.1.1 で説明したとおり高次元への射影にコストがかかる、などの問題がある。このことは、ほぼ自明であると思われるが、実際に将棋盤のハッシュ値を入力ベクトルに入れて学習させるなどという乱暴なことも行ってみて、分類がまったくできないことも確認した。このようなことから、ここでは将棋の知識を元に特徴量ベクトルを作ることにした。

SVM への入力としての特徴量には Grimbergen ら [8] の用いた 161 要素の特徴量を元に、その特徴量から追加、削除、統合した 129 要素のベクトルを用いた。これらの 129 要素のベクトルとは、次のような特徴を攻め手、守り手両方について取り、あわせたものである。

- 王について
  - 王の絶対位置 (行、列)
  - 両玉のマンハッタン距離
- 王の周り 8 枡、王の前 14 枡、後 10 枡について
  - 端
  - 駒がある
  - 敵に有利または自分に有利 (利きの数が多いほうを有利とした)
  - 王の逃げ道の数
  - 守り駒 (王側の駒)、攻め駒 (敵側の駒)
  - ピン[2]されている駒
- 持ち駒について
  - 持ち駒の総数
  - それぞれの駒の数
- 王への攻撃の可能性について
  - 王への利き (間接利き含む) ができる可能性のある指し駒の数と動ける位置の数
  - 王への利き (間接利き含む) ができる可能性のある持ち駒の置ける位置の数

特徴量は、もともと将棋盤の表現の中に持っているもの、もしくはそれらの和、差のみで得られる、ほとんど計算を行うことなく取り出せるものを用いている。例外として、王への攻撃の可能性に関しては、一段先の探索を行っていることに対応し、将棋盤を一度見る必要がある。

このような特徴量の抽出は経験的なものであるので、見落としがあることはさけられないが、分類器の性能をよりよいするために次のような事前実験を行った。

特徴量の抽出について 本節の始めにも書いたが、実際に将棋盤のハッシュ値を入力ベクトルに入れて学習させる事を行ったが、まったく分類はできなかった。

王の周りの特徴量について 王の周りについて前後の特徴量を分割したのは、将棋は左右には対称であるが、上下には非対称であるためである。分割した場合は、分割しない場合に比べて正解率が 3%程度改善した。

また、すべての枡を統合せずに別々の特徴量として用いることも考えられるが、このような場合に比べて前後で統一した場合は 1%改善した。

また、王の周り 49 枡について見るということも行い 0.5%程度向上することを確認したが、その特徴量を抽出するのに時間がかかるため今回は特徴量からはずすことにした。

[2]動かすと味方の王に王手がかかってしまう動けない駒。

王への攻撃の可能性について 王への攻撃の可能性に関しては、静的に一段先の探索を行っていることに対応し、将棋盤を一度見る必要がある。このように他の特徴量に比べて重いものであるにもかかわらず、このような要素を入れたのは、入れることで正解率が 2%程度向上したためである。

進行度を考慮した要素について [26] のような局面の進行度を考慮した要素を取り入れた場合については正解率が 1%程度下がり、ほとんど効果が無いことが確認できた。

無駄な特徴量の削減について 今回挙げているこの 129 要素の中にも影響が大きいもの、小さいものがあった。影響が大きい、小さいということは、式 (2.14) より、その係数が大きい、小さいにそのまま対応するので、小さいものを取り除くことで速度を改善させることも考えられるが、極端に小さいものはみられなかったため取り除くことはしていない。また、極端に小さくはないが影響の小さいものを取り除くため、全体の下から 10%程度 (係数が 0.01 以下のもの) を取り除いた場合、正解率は 1%ほど低下した。

このように、特徴量としてどの特徴をとるかということは非常に重要な問題であり、上記の特徴量にも改善の余地がある。

3.2.3 データについて

データは次のように作成した 40000 局面を用いた。

1. プロの対局 4000 局それぞれについて終局までの最後の 10 局面を取り出す。
2. 取り出した 40000 局面それぞれについて浅く探索し、探索中に出てきたノードをランダムに 1 つずつ取り出す。

最後の 10 局面を取り出したのは、そこが詰み判定が最も有用とされる場面であり、この範囲で学習したものは、実際の詰み判定でも有用であると考えられるためである。また、それぞれに関して浅い探索を行ったのは、探索中には頻繁に出てくるが、対局中にはあまり出てくることのないような (機械的な) 局面も分類器に判別させたいと考えたためである。

上記のようにして取り出した 40000 局面の中には詰みの局面が 10159 個あった。この詰みの割合は、終局に近いところ以外では、実際の将棋に起こりうる局面全体に含まれる詰みの割合よりも大きい。そのため実際に利用する際には詰みと認識してしまう可能性が大きくなることを考慮する必要がある。このような問題が考えられるにもかかわらず、今回のような詰みに偏ったデータを用いたのは、主には現在の計算環境のためである。

実際に学習させるデータとしては、実際の将棋の傾向にあったものを用いるのが望ましい。しかし、事前実験において、対局全体からランダムに 40000 局面取り出して (詰みの局面が 120 局面で) 学習させたところ、このデータで学習した SVM に対してどちらのデータに関しても正解率が低いという結果が得られた。このような結果になったのは 120 局面という数が将棋の詰みの学習の要素数としては少なく、詰みの知識をほとんど得られなかったためだと考えられる。ここで、学習するデータ数を増やすことが考えられるが、将棋の詰みの学習に必要な局面数は非常に大きいことは明らか

で時間の制約上現在の環境では困難である。このようなことから詰みの要素を増やすことが必要となり、今回のようなデータを用いた。

3.2.4 実験環境

実験は CPU Xeon 2.40GHz dual, メモリ 2GB の環境で行った。また、SVM は SVM のライブラリである LIBSVM[5] を線形専用に改良したものを使用した。

3.3 分類器の性能

分類器の性能については表 3.1, 3.2 のような結果が得られた。この結果は 3.2.3 に示した 40000 局面のデータを 4-fold cross validation で (つまり 30000 局面を学習データに残り 10000 データをテストデータにして 4 回評価を行うことで) 評価した結果である。また、計算時間を示しておくと、(ラベル付の)30000 局面の学習に 3 時間近くかかったのに対し、分類には 1 局面あたり 6.99μsec で終わるという結果がえられた。この値については、特徴量の一部は先読みに当たっての局面の更新時に差分計算をしているため、特徴量を新たに算出する場合よりは若干早くなっている。

この正解率は、学習データに関して詰探索ではあり得ない false positive(不詰の局面を詰みと判断すること) はあるものの PDS で 244 ノード探索した場合と比較したときの値とほぼ同じ結果となっている。244 ノードは『激指』で実際に探索する際、葉ノードから 7 段目を探索するときに使う制限とほぼ等しく、より深いところでは『激指』の詰探索の代用に使うことも考えられる。実際にこのことを考えて、単純に深いところで詰探索の代用としてこの分類器を用いることを試みたが、ほとんどの場合において、探索時間が増えてしまうという結果になった。この false positive、つまり詰むと分類したのに詰まなかったという状況が、実際の詰探索には起こりえない状況であり、ゲーム木自体に少なからず影響を及ぼしてしまったためであると考えられる。

また、分類には 1 局面あたり 6.99μsec でできるという結果が得られたが、PDS が 142 ノード/msec 程度であることから、この分類器は PDS で約 1 ノード探索する時間で分類できており、この分類器は実用にも十分に耐える計算時間で分類ができているということがわかる。

表 3.1: 分類器による 40000 局面の分類結果

	詰み	不詰	合計
詰みと分類された数	6414	1865	8279
不詰と分類された数	3745	27976	31721
合計	10159	29841	40000

表 3.2: 分類器の正解率

	Precision	Recall	f1
詰み	0.775	0.631	0.696
不詰	0.882	0.938	0.909

第4章　詰み予測の応用

本章では、3で得られた詰みの予測を行う分類器に関する応用の可能性について4.1で述べた後、その応用例として4.2でPDSの探索ノード数制限の制御について、4.3でPDSの評価関数への適用について、は4.4でPDSの証明数、反証数の上限の制御について、それぞれ述べる。

4.1　SVMによる詰みの予測の応用

本研究で作成した分類器には次のような多くの応用が考えられる。

1. 通常の探索と詰みチェックアルゴリズムの計算時間の分配を調整すること。
2. 分類器をPDSなどの詰探索アルゴリズムの探索順序制御 (move ordering) に適用すること。
3. 分類器をPDSなどの詰探索アルゴリズムの評価関数に利用すること。
4. 分類器をPDSなどの詰探索アルゴリズムのPN, DNの上限の制御に用いること。
5. 他の詰探索アルゴリズムの代用として利用すること。
6. 詰探索以外の場所 (評価関数など) に利用すること。

本研究で実験・評価を行っているのは主に1, 3, 4についてである。2, 5については、実験の結果ほとんど効果がないことが分かった[21]。1については4.2で、3については4.3で、4については4.4でそれぞれ説明する。

4.2 PDS の探索ノード数制限の制御

本節では、分類器の応用の 1 つとして、分類器による詰み予測の結果を元に詰探索の探索ノード数制限を制御する手法について 4.2.1 で述べ、その実験方法、実験結果をそれぞれ 4.2.2、4.2.3 に示す。

4.2.1 PDS の探索ノード数制限の制御

2.2.3 に示したとおり、指将棋プログラム内では探索中の全てのノードで探索ノード数を制限しながら詰探索を行う。この探索ノード数制限は、深さが同じところでは、その詰みやすさ、その証明木の大きさに関わらず同じである。この探索ノード数制限を詰みに関する情報で変化させることができれば、より柔軟で効率的な探索が可能となることが考えられる。このようなことから、図 4.1 の※に示したように、この詰探索の探索ノード数の制限を分類器による詰み予測の結果に応じて制御することを試みた。

4.2.2 実験方法

ここでは、実験方法として、図 4.1 の探索ノード数の制限の制御をどのようにして行ったかについて述べる。

まず、分類器が不詰であると判断したときは探索ノード数の上限を減らすという方法をとった。これは、3.2.3 の 40000 局面から学習して得られた分類器には不詰であると判断した場合に次の 2 つの傾向があったためである。

- 図 4.2 に示したように、90%以上の正解率で (特に探索に用いられるような数千ノード程度以下の少ないノード数制限で探索した結果との比較では 98%以上の正解率で) 不詰もしくは不明である。
- 探索に用いられるような数千ノード程度以下の少ないノード数制限で探索したときには、探索しても不明という結果になることが多い。

また、分類器が詰みと判断したときは、不詰と判断された場合の探索時間を減らした分、その探索ノード数の上限を増やすことにした。これは

- 今回得られた分類器が詰みであると判断した場合に 77.5%が詰みであること
- 詰みを早めに見つけることでより正確な探索ができると考えられること

からこのようにした。

また、この制御を行う際、分離超平面からの距離が詰む詰まないの予測の精度と相関が高いと考えられ、これをパラメータとする関数を用いることで、PDS の探索ノード数制限の制御を試みた。この関数として、図 4.3 に示した次の 3 つの関数を用いた。

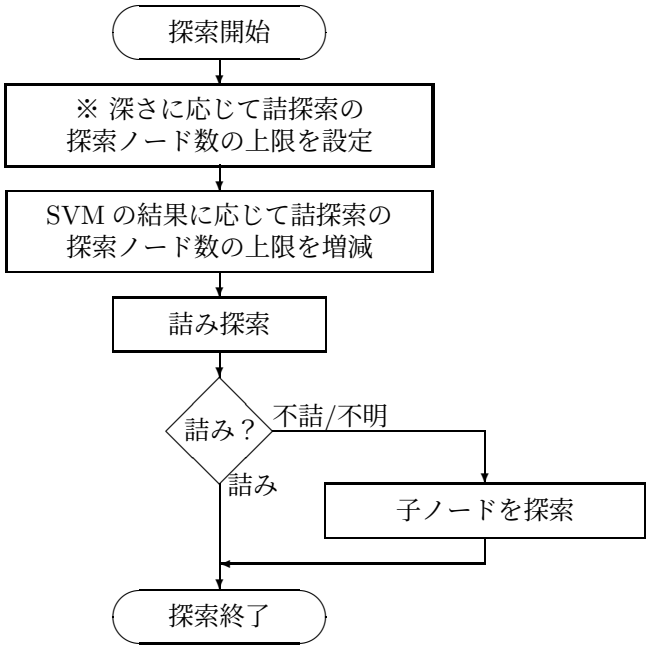

図 4.1: PDS の探索ノード数制限の制御

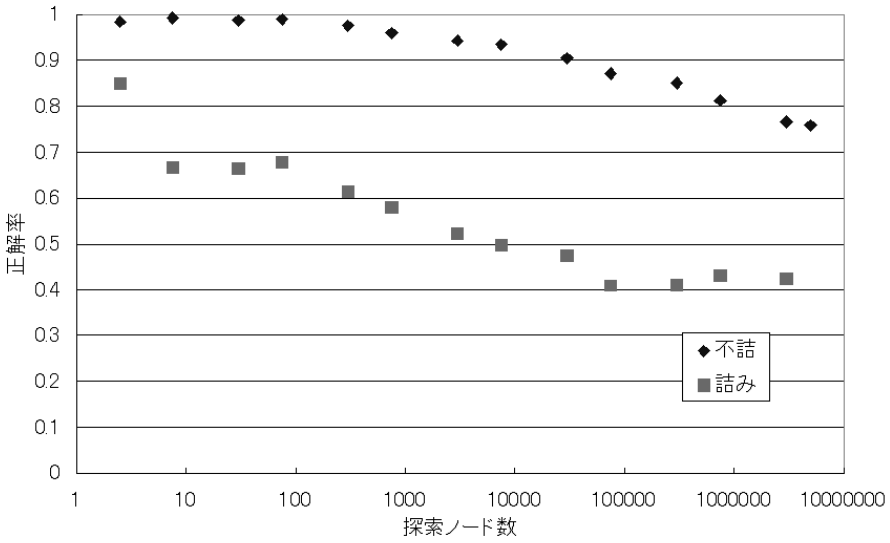

図 4.2: 局面の PDS 探索ノード数に対する正解率の変化

- 分離超平面からの距離にあわせて線形に変化する関数

$$f(x) = \alpha x + 0.5 \tag{4.1}$$

- 分離超平面からの距離を基にしたシグモイド関数

$$f(x) = \frac{1}{1 + exp(-x)} \tag{4.2}$$

- 分離超平面を境界に 2 値をとる階段関数

$$f(x) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases} \tag{4.3}$$

線形の関数については図 4.3 のように学習データの中で正例の中で分離超平面からの距離が最も大きいものが等倍、分離超平面が 1/2 倍になるようにして決定し、式 (4.1) における α を 0.05 とした。また、探索全体の詰探索時間の比率が変わらないようにするために関数全体を定数倍した。

また、図 4.4 に示したのは学習に用いた局面を基にした超平面からの距離ごとの局面の数であり、図 4.5, 4.6 に示したのはそれぞれ PDS の探索ノード数の上限を 250, 100000 としたときの超平面からの距離ごとの平均探索ノード数である。本実験において、どのように探索全体の詰探索時間の比率が変わらないようにしたかというと、探索ノード数 (図 4.4 に図 4.5, 4.6 のような平均探索ノード数を掛けたもの) に関数による制限をかけ、その探索ノード数が同じになるように探索ノード数制限ごとに倍率を調整することを行った。

4.2.3 実験結果

探索深さを固定した上で、探索全体の詰探索時間の比率が変わらないようにして、制御を行わないもの (オリジナル) と 3.2.3 の 40000 局面から学習して得られた分類器を用いて制御を行ったものを対戦をさせた場合の結果を表 4.1 に示した。そして、この中のシグモイド関数を用いた場合についてその 1 ゲームごとの探索時間の違いを図 4.7 に示した。

ここで、n-sigma というのは勝敗を正規分布と仮定したときのシグマ値である。

表 4.1 を見ると、いずれの関数についても探索精度を変えずに、ゲーム全体の探索時間を大きく削減していることがわかる。図 4.7 では特にその探索時間の違いが見て取れ、ほとんど全てのゲームにおいて、オリジナルのほうの探索時間のほうが大きくなっていることが分かる。階段関数については、他の結果に比べてその削減量が少ないが、これは不詰と判断されたときに全く探索を行わなかったために、その不詰と判断された局面が実際には間違いですぐに詰みとわかるような局面であった場合の無駄な探索が他の関数に比べて多かったためであろうと考えられる。他の 2 つの関数はそれに比べて、探索時間がより多く削減されており SVM の超平面からの距離には、分類以上の意味合いがあり、その探索ノード数の上限との何らかの相関があることの裏づけの 1 つとなったといえる。この最も削減されたシグモイド関数を用いた手法について、対局をその手数について単純に 3 分割して初盤、中盤、終盤と分けた場合それぞれについてのゲーム全体の探索時間の比は 1.01, 0.635, 0.572

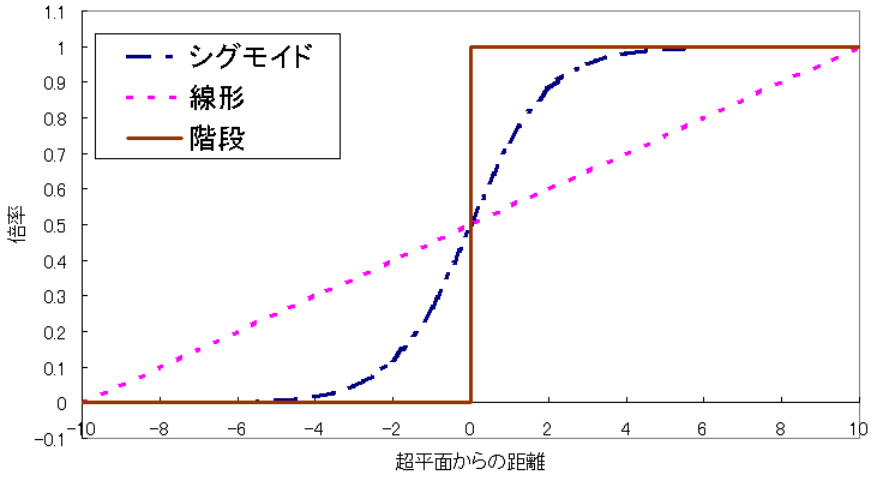

図 4.3: 分離超平面からの距離と予測の精度を表現する関数

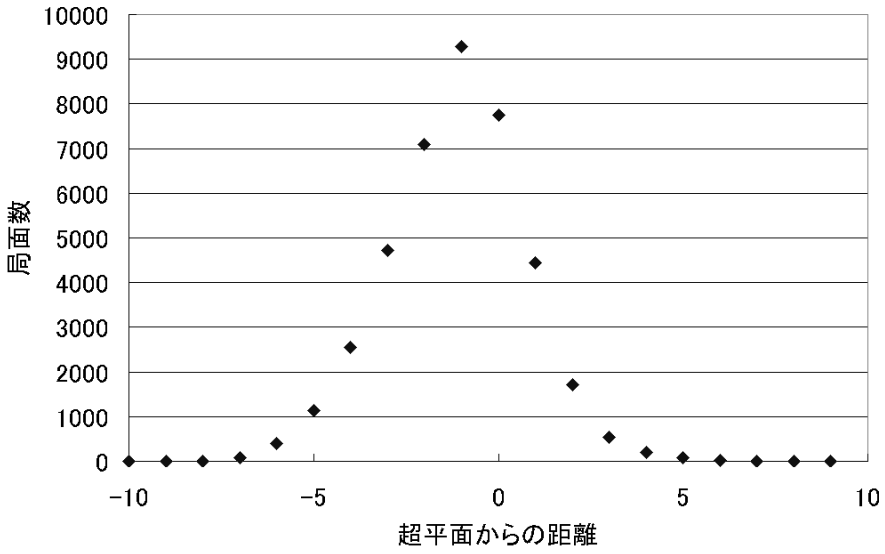

図 4.4: 超平面からの距離ごとの局面数

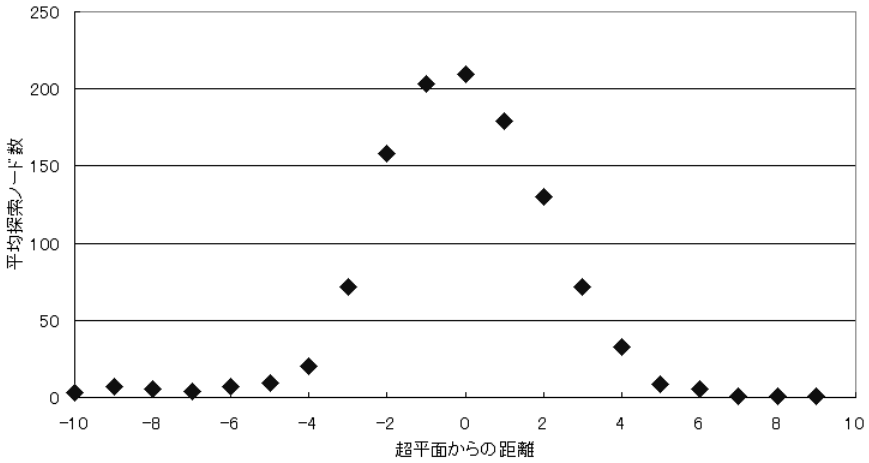

図 4.5: 超平面からの距離ごとの上限 250 ノードのときの平均探索ノード数

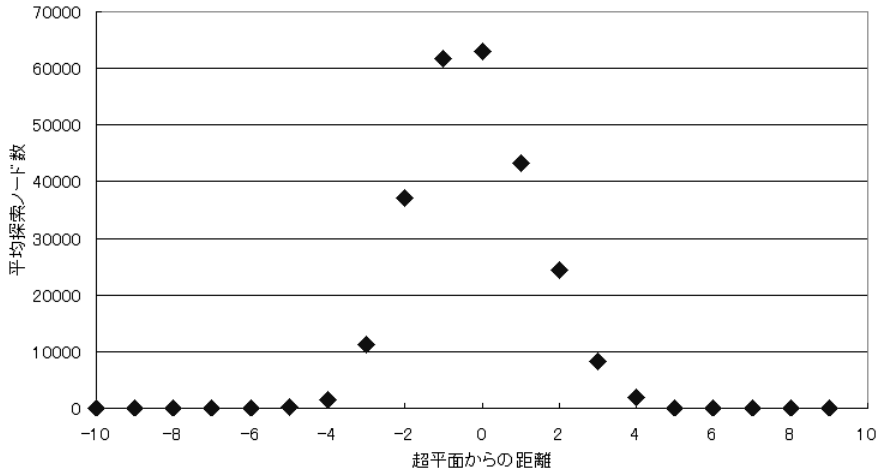

図 4.6: 超平面からの距離ごとの上限 100000 ノードのときの平均探索ノード数

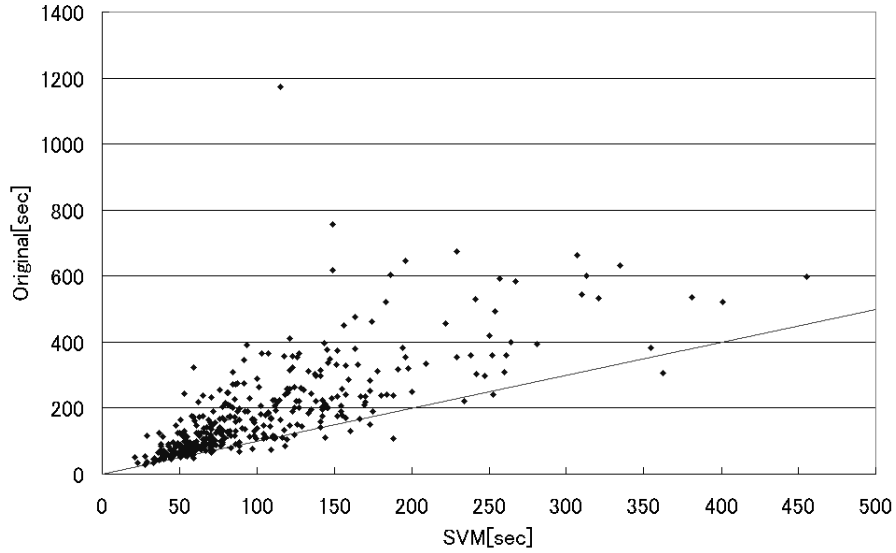

図 4.7: シグモイド関数を用いた場合のオリジナルと分類器を用いたものの探索時間

となった。このことから、この手法は詰みチェックが有用な中盤、終盤において特に有効だということがわかる。また、統計的に有意といえないまでも、全ての結果において勝率が 5 割を超えている。このような結果になったのは詰探索が正確になったためであると考えられる。

次にこのゲーム全体の探索時間が最も削減されたシグモイド関数を用いた手法についてその関数にかける定数の大きさ (探索全体の詰探索時間の比率が変わらないものを 1) を変化させた場合の勝率と探索時間の変化を表 4.2 に示した。この結果を見ると、勝率は 2 倍のときに最も大きくなっており、この時の勝率は統計的に有意であるということができる。また、3/4 倍のときその探索全体の探索時間の削減量は最大になっておりその強さを変えずに 61.9%まで削減、つまり約 4 割の時間を削減することができている。1/2 倍のときに、勝率が下がって、他の傾向と異なり探索時間が増えているが、これは PDS の探索ノード数制限がオリジナルよりも常に小さい値になっており、詰探索が不正確になったためであると考えられる。

表 4.1: 制御をしない『激指』と 1000 回対戦した時の勝率と探索時間

	線形	シグモイド	階段
勝率	0.545	0.534	0.503
(n-sigma)	(2.77)	(2.11)	(0.16)
ゲーム全体の探索時間の比 (オリジナルを 1)	0.762	0.656	0.833

表 4.2: シグモイド関数を用いた場合の定数を変化させた場合の勝率と探索時間の変化

	1/2	3/4	1	2	4
勝率	0.448	0.503	0.534	0.553	0.528
(n-sigma)	(-3.23)	(0.16)	(2.11)	(3.26)	(1.73)
探索時間の比	0.907	0.619	0.656	0.768	0.900

4.3 PDS の評価関数への適用

本節では、分類器の応用の 1 つとして、分類器による詰み予測の結果を PDS の評価関数に用いる手法について 4.3.1 で述べ、その実験方法、実験結果についてそれぞれ 4.3.2、4.3.3 に示す。

4.3.1 PDS の評価関数への適用

2.2.2 に示したとおり、PDS ではそのノードが探索されていない (不明である) 場合、その証明数、反証数を 1 に設定する。この不明の際に、ノードの証明数と反証数を予測することができればより効率的に探索できると考えられる。実際に PDS に似たアルゴリズムである df-pn アルゴリズムには、そのノードの証明数と反証数を予測する評価関数を用いて探索を効率化する df-pn+アルゴリズムが提案されており、元の df-pn アルゴリズムよりもよい結果を示している [13][25]。このようなことから、3.2.3 の 40000 局面から学習して得られた分類器をこの局面の証明数、反証数を予測する評価関数として用いる事を行った。

4.3.2 実験方法

ここでは実験方法としてどのように評価関数を構築するのかについて説明した後、その評価関数のパラメータを決める予備実験について述べる。

評価関数は次のような動作をするものとして構築した。まず、証明数に関しては、それを大きくすることは詰探索を後回しにすることに対応する。そのため、詰みと判断されたときは証明数を 1 とし、不詰と判断されたときは $1+\alpha$(αは 0 より大きい整数) とした。同様に、反証数に関しては、それを大きくすることは不詰探索を後回しにすることに対応する。そのため、詰みと判断されたときは反証数を $1+\alpha$ とし、不詰と判断されたときは 1 とした。ここで $1+\alpha$ として説明しているのは、この証明数、反証数の予測値をいくつにするのが適切であるかということは分かっていないためである。表 4.3 に評価関数の設定について示した。

この予測値は次のようにして決定した。まず、この予測値には大きいほど探索を後回しにする効果が大きくなるという性質がある。この性質により、分類器が信頼できるものであれば予測値を大きくすることは価値があるが、[21] において最良優先探索を行った場合に間違ったノードを展開しつづけるということが起きたことから、この分類器は似たような局面については同じように間違うことが考えられる。そのため、この予測値をあまり大きくしすぎるとその間違いの影響が大きくなってしまうことは避けられず、あまり大きくしすぎないほうがよいと考えられる。また、根本的な問題ではないが、その値をあまり大きくしすぎると、巨大な探索を行ったときにルートノードの証明数、反証数の値が大きくなりすぎ、証明数、反証数を保存するハッシュ表のエントリのサイズを大きくしなければならない、という望ましくない問題も起こる。このようなことから、この値を極端に大きくするようなことは避けつつ、値を変化させて、実際に探索してみることで決定した。

また、今回は 4.2 において、超平面からの距離には意味があることが分かっているので、4.2 において最も成績のよかったシグモイド関数を用いて距離が大きいほど予測値が大きくなるようにした

場合についても表 4.4 のようにして評価関数を作成し、実験を行った。

また、探索深さが深いところで呼べるほどこの分類器は高速ではないため、探索深さの深いところでは呼び出さないようにする必要がある。これは、探索深さが深すぎるところまで呼び出すと、[21] において最良優先探索を行った場合に間違ったノードを展開しつづけるということが起きたことから、間違った局面の子ノードは間違いやすく、大きな間違いを犯してしまうことがあることが考えられるため、それを避けるという目的もある。

ここで、この α を決めるために行った予備実験とその結果について述べる。

まず、1 つ目の実験として、3.2.3 と同様にして作った 100 局面について 1000000 ノードを上限として、α を変化させて探索した場合の実験結果を図 4.8 に示した。図 4.8 の探索ノード数は全ての局面の探索ノード数の総和、点線は評価関数を利用しない場合の探索ノード数である。また、階段というのは表 4.3 のように超平面からの距離を考慮せずに α+1 を予測値としたものであり、シグモイドというのは表 4.4 のように超平面からの距離をシグモイド関数を用いて平滑化しその値に α を掛けて 1 を足したものを予測値としたものである。ここで、図 4.8 において 2 つの関数を比較する際に注意しなければならないのは、このように予測値は階段ではその値を用いているのに対し、シグモイドではその値をシグモイド関数を使って小さくしているため、階段の α はシグモイドの α の約 2 倍の効果があるということである。この際、深すぎるところでは分類器を呼び出さないようにするために、深さの打ち切りを 4 として、それ以上の深さでは分類器を呼び出さず不明の場合の証明数、反証数の値を 1 とするようにして実験を行った。結果を見ると、特に α が大きくなるほど探索ノード数が大きくなっており、α は小さな値をとることが望ましいということが分かる。しかし、α を小さく取ればよいのかというとそういったわけでもなく、全ての α において評価関数を利用しない場合と同じかそれよりも探索ノード数が多くなっている。このようなことがおきたのは、この予備実験の方法では、ある深さまではその証明数、反証数は (最大で)1+α であるのに対し、その深さを超えるとその証明数、反証数の値は 1 になり、この打ち切り深さについて証明数、反証数が不連続になっているためであると考えられる。

このため、2 つ目の実験として、この深さについての証明数、反証数の不連続性をなくすために、深さに合わせてその証明数、反証数の予測値の最大値を小さくするという方法を取って、同じ 100 局面について 1000000 ノードを上限として実験を行った。この予測値の最大値を小さくする手法として、超平面からの距離を考慮しない場合については表 4.5 のように α - depth + 1 を予測値として、超平面からの距離を考慮する場合については表 4.6 のように α - depth に超平面からの距離をシグモイド関数を用いて平滑化したものを掛け、1 を足したものを予測値として探索を行った。この場合の結果を図 4.9 に示した。これらの評価関数は、α が depth 以下となったときには評価値を 1 とすることで、深すぎるところでは呼び出さないようにしている。この結果を見ると、シグモイド関数を用いて平滑化した場合では全ての α において評価関数を利用しない場合と同じかそれよりも探索ノード数が多くなっている。一方で平滑化しなかった場合については α が 5 のときに最小になっており、このとき評価関数を利用しない場合よりも探索ノード数が小さくなっている。

これら 2 つの予備実験の結果から、$\alpha = 5$ として、超平面からの距離を考慮せず証明数、反証数の予測値を深さにあわせて小さくする手法を用いることにした。

表 4.3: 超平面からの距離を考慮しない場合の PDS の評価関数

分類器の評価	詰み	不詰
証明数	1	$1+\alpha$
反証数	$1+\alpha$	1

表 4.4: 超平面からの距離 distance を考慮した場合の PDS の評価関数

分類器の評価	詰み	不詰
証明数	1	$1+\frac{\alpha}{1+exp(-\text{distance})}$
反証数	$1+\frac{\alpha}{1+exp(\text{distance})}$	1

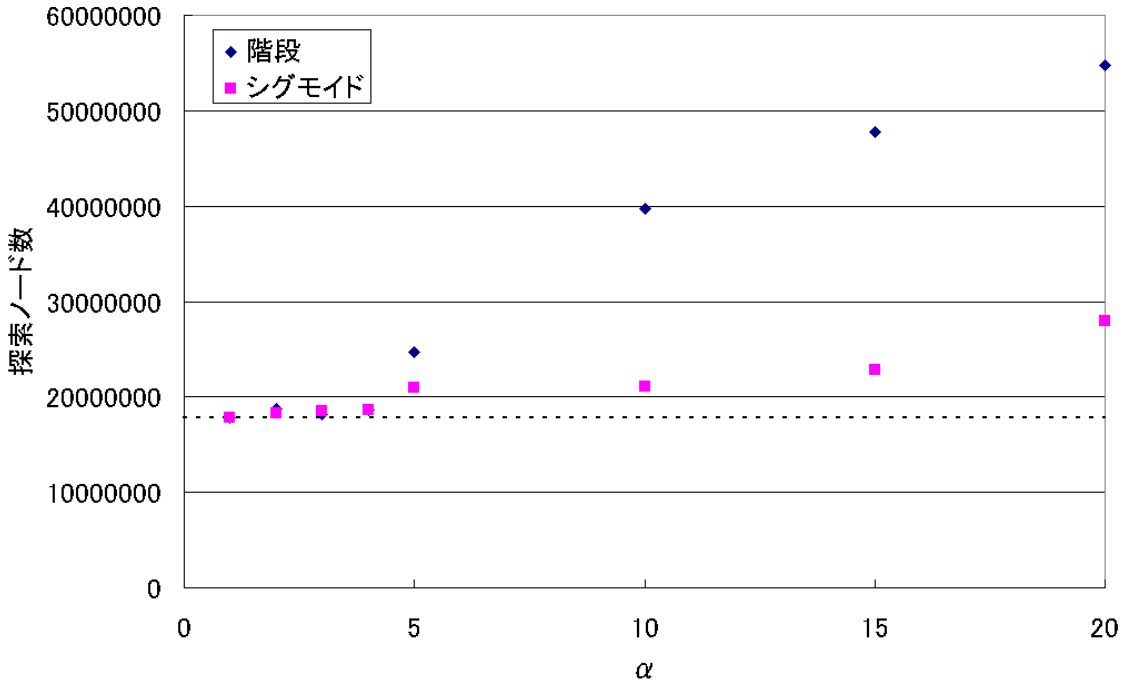

図 4.8: SVM による予測を評価関数として PDS で 1000000 ノード制限で 100 局面探索したときの探索ノード数

表 4.5: 深さ depth を考慮し、超平面からの距離を考慮しない場合の PDS の評価関数

分類器の評価	詰み	不詰
証明数	1	$1+\alpha$ - depth ($\alpha >$ depth) 1 ($\alpha \leq$ depth)
反証数	$1+\alpha$ - depth ($\alpha >$ depth) 1 ($\alpha \leq$ depth)	1

表 4.6: 深さ depth、超平面からの距離 distance を考慮した場合の PDS の評価関数

分類器の評価	詰み	不詰
証明数	1	$1+\frac{\alpha-\text{depth}}{1+exp(-\text{distance})}$ ($\alpha >$ depth) 1 ($\alpha \leq$ depth)
反証数	$1+\frac{\alpha-\text{depth}}{1+exp(\text{distance})}$ ($\alpha >$ depth) 1 ($\alpha \leq$ depth)	1

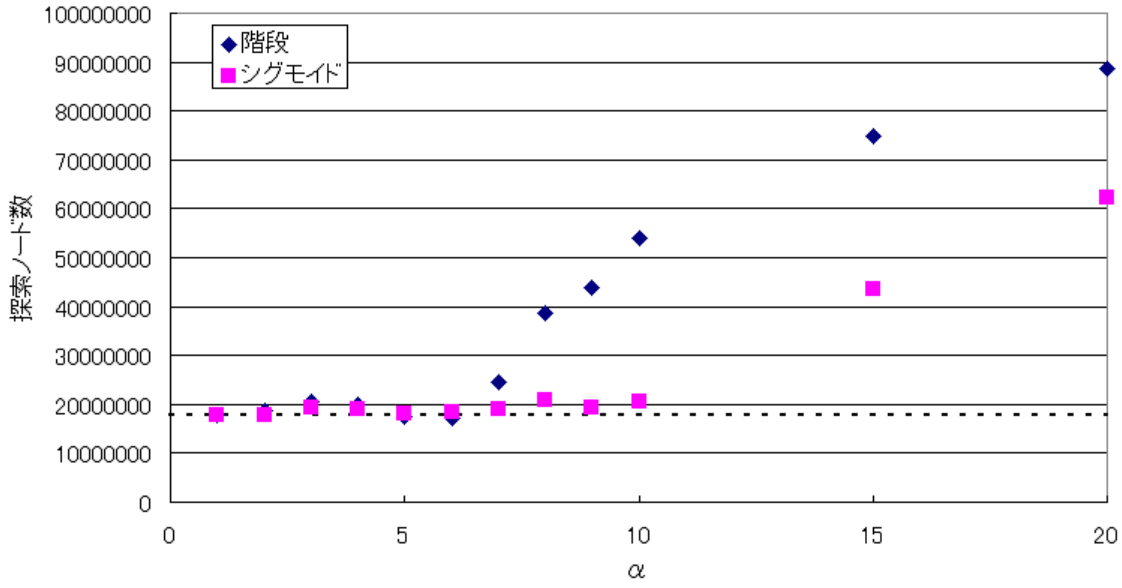

図 4.9: 予測値を深さに合わせて減少させた時の 1000000 ノード制限で 100 局面探索したときの探索ノード数

4.3.3 実験結果

3.2.3 と同様にして作った 10000 局面について 1000000 ノードを上限として、評価関数を利用しない PDS と評価関数を利用する PDS を使って詰探索を行うことで実験を行った。表 4.7 に探索ノード数の和と探索時間の和、表 4.8 にその中で両者の探索が不明でないもののの探索ノード数の和と探索時間の和、表 4.9 に探索の解答結果、図 4.10 に探索ノード数のグラフ表示をそれぞれ載せた。

この結果から、評価関数を利用した場合に比べ少ないノード数で解ける問題は若干多くなっている。また、探索ノード数も減少しており、特に両者が不明でないとき、その探索ノード数が 14.6%程度削減されている。その一方で、探索時間については全体で 1%程度増加している。このように探索時間が増加したのは分類器の呼び出しによる影響であると考えられる。

また、[25] のように詰探索の探索ノード数は最小の証明木の 10 倍から 100 倍であるにもかかわらず、このようにノード数があまり削減されない結果となったのは、この分類器の予測と証明数、反証数との相関がそれほど高くないことが考えられる。

このような結果からよりこの分類器の予測を有効に使うための方法としては次の二つが考えられる。

1. 確率を使って証明数、反証数を制御する方法。
2. 証明数、反証数の上限のみを制御する方法。(4.4)

ここでは、確率による制御について説明する。今回は AND ノード、OR ノードを同等に扱っているが、AND ノードでは全ての子ノードが証明される必要があるが、OR ノードでは 1 つの子ノードが証明されればよいという違いがある。このようなことから、この分類器の予測を詰みの確率として考えた場合には、表 4.10 のような方法で決定する、もしくは [1] のような Bayesian approach を用いるなど、証明数、反証数の枠組みにとらわれない方法で証明数、反証数を制御することを行う必要があると考えられる。ただし、このような場合には今回以上に計算時間がかかることが考えられる。

表 4.7: 評価関数を利用した時の探索ノード数の和と探索時間の和

	探索ノード数	探索時間 [sec]
評価関数を利用しない場合	3374215066	17483
評価関数を利用した場合	3322195669	18130

表 4.8: 探索が不明でない時の評価関数を利用した探索ノード数の和と探索時間の和

	探索ノード数	探索時間 [sec]
評価関数を利用しない場合	355467384	881
評価関数を利用した場合	303447858	880

表 4.9: 評価関数を利用した時の解答結果

	詰み	不詰	不明
評価関数を利用しない場合	2041	4809	3150
評価関数を利用した場合	2048	4855	3097

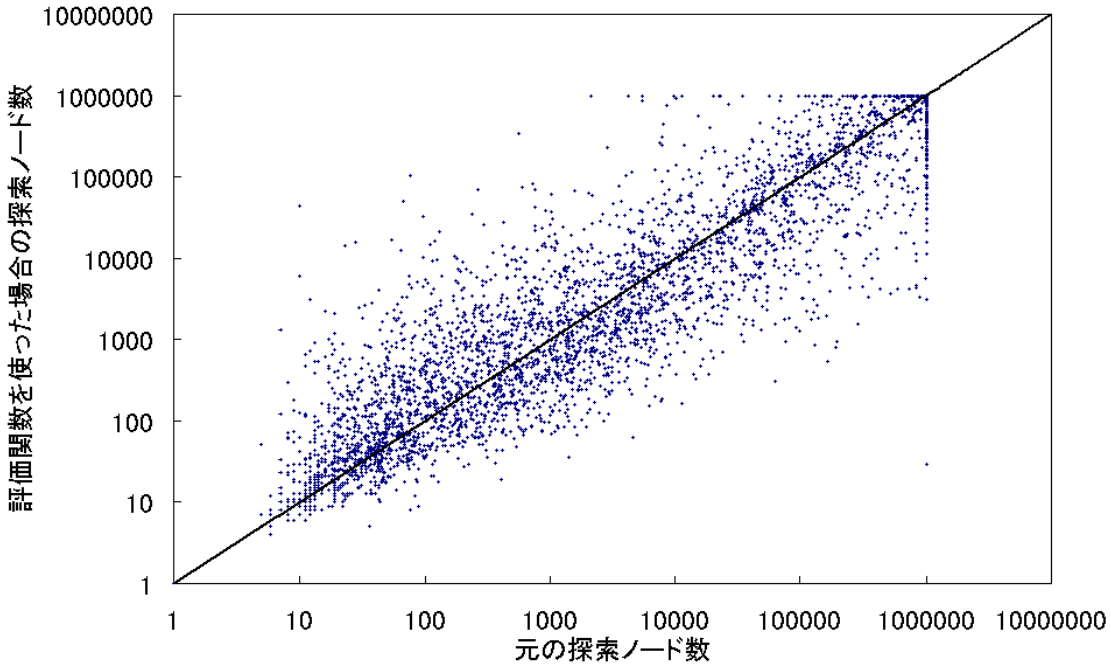

図 4.10: 評価関数の利用による 1000000 ノード制限で 10000 局面探索したときの探索ノード数の違い

表 4.10: 子ノードの確率 p_i によるノードの確率 p の決定方法

仮定 / ノードの種類	子ノードが独立	子ノードが従属
AND ノード	$\prod_i p_i$	$min(p_i)$
OR ノード	$1 - \{\prod_i (1 - p_i)\}$	$max(p_i)$

4.4 PDS の証明数、反証数の上限の制御

本節では、分類器の応用の 1 つとして、分類器による詰み予測の結果を PDS の証明数、反証数の上限の制御に用いる手法について 4.4.1 で述べ、その実験方法、実験結果についてそれぞれ 4.4.2、4.4.3 に示す。

4.4.1 PDS の証明数、反証数の上限の制御

今回の詰み予測は 4.3.1 の結果の通り、詰探索の探索ノード数の上限と相関があることが考えられる。

PDS のような手法においては、詰探索の探索ノード数の上限と同じ性質を持つと考えられるものとしては多重反復深化を行う際の証明数、反証数の上限が考えられ、通常の PDS では 2.2.2 に示したとおり、その証明数、反証数の上限を少ないほうから 1 ずつ増加させる方法を取るが、その増加の順序や値を制御することにより、高速に探索を行うことができると考えられる。

このようなことから、PDS の証明数、反証数の上限の増分を詰みの予測の結果に応じて制御することを行った。

4.4.2 実験方法

ここでは、PDS の証明数、反証数の上限の増分をどのように制御したのかについて説明する。

その上限の増分の制御については基本的には次のようなことを行った。証明数の上限を増加させることは詰探索を多く行うことに対応する。そのため、詰みと判断されたときは証明数の上限の増分を $1+\alpha$(α は 0 より大きい整数) とした。同様に反証数の上限を増加させることは不詰探索を多く行うことに対応する。そのため、不詰と判断されたときは反証数の上限の増分を $1+\alpha$(α は 0 より大きい整数) とした。また、証明数、反証数の増分の制御の影響を維持するため、証明数、反証数の上限の少ないほうから増加させるかわりに、証明数、反証数をそれぞれの増分で割ったものが小さいほうから増加させることにした。このようなことを行うことにより、詰探索、不詰探索にほぼそれぞれの増分の比だけ探索量を割り当てを行うことができると考えられる。

ここで、この α という値は、4.3.2 と同様、実験を行いながら決定した。この事前実験においては 4.3.2 と同様に、次のようなことを考慮する必要があると考えられる。

- 超平面からの距離
- 深さが深すぎるところでこの分類器を使うのはコストが大きく、また、その予測の失敗が連続して起きる可能性があること
- その増分が不連続になることが影響を及ぼす可能性があること

これらのことを考慮し、次の 2 つの予備実験を行った。それぞれについてその増分を超平面からの距離をシグモイド関数を用いて平滑化したものを用いて制御することも行った。

- 深さを一定にした実験
- 深さに合わせてその増分を減少させる実験

まず、深さを一定にした実験について説明する。深さを 4 に固定して、証明数、反証数の増分として表 4.11、4.12 に示したものを用いて実験を行った。評価としては、4.3.2 で用いた 100 局面について 1000000 ノードを上限として、α を変化させて探索することにした。この実験結果を図 4.11 に示した。ここで、点線は増分を変化させないときの探索量、表 4.11 が階段、表 4.12 がシグモイドにそれぞれ対応する。この実験については、深さについても変化させる必要があると考えられるが、実験結果で最小となった α を 6 として超平面からの距離をシグモイド関数を用いて平滑化したものを用いて制御し、その深さを変化させたところ、この実験で用いた深さ 4 において最小となったため、深さを変化させることはしなかった。

次に深さに合わせてその増分を減少させる実験について説明する。こちらについては、証明数、反証数の増分としては表 4.13、4.14 に示したものを用いて実験を行った。評価としては、上記の実験と同様の 100 局面について 1000000 ノードを上限として、α を変化させて探索することにした。この実験結果を図 4.12 に示した。ここで、点線は増分を変化させないときの探索量、表 4.13 が階段、表 4.14 がシグモイドにそれぞれ対応する。

いずれの実験の結果についても、1 から 10 のほとんどの α について元の探索に比べて探索ノード数を削減できていることが分かる。

これらの予備実験の結果から、α を 7 として、深さに合わせてその増分を減少させ、その増分を超平面からの距離をシグモイド関数を用いて平滑化したものを用いて制御することにした。

4.4.3 実験結果

4.3.3 で用いた 10000 局面について 1000000 ノードを上限として、証明数、反証数の上限を制御を行わない PDS と制御を行う PDS を使って詰探索を行うことで実験を行った。表**??**に探索ノード数の和と探索時間の和、表 4.16 にその中で両者の探索が不明でないもののの探索ノード数の和と探索時間の和、表 4.17 に探索の解答結果、図 4.13 に探索ノード数のグラフ表示をそれぞれ載せた。

この結果を見ると、不詰に関して多く正解を出しており、詰みに関してはその解答数が減っている。これは、この詰みの予測が詰みに関しては不詰に比べて間違えることが多く、探索が不詰に傾いたためであると考えられ、分離超平面の位置を調整することで解決できるのではないかと考えられる。探索全体では探索ノード数 12.2%、探索時間 8.44%削減されており、特に両者が不明でないとき、その探索ノード数が 85.7%、探索時間が 84.5%程度削減されている。4.3.3 の結果に比べて、この結果は非常に良い結果であり、この詰みの予測がうまく働いた結果であると考えられる。このことから、4.3.1 の結果と同様に詰探索の探索ノード数の上限と詰みの予測は相関があることが考えられる。

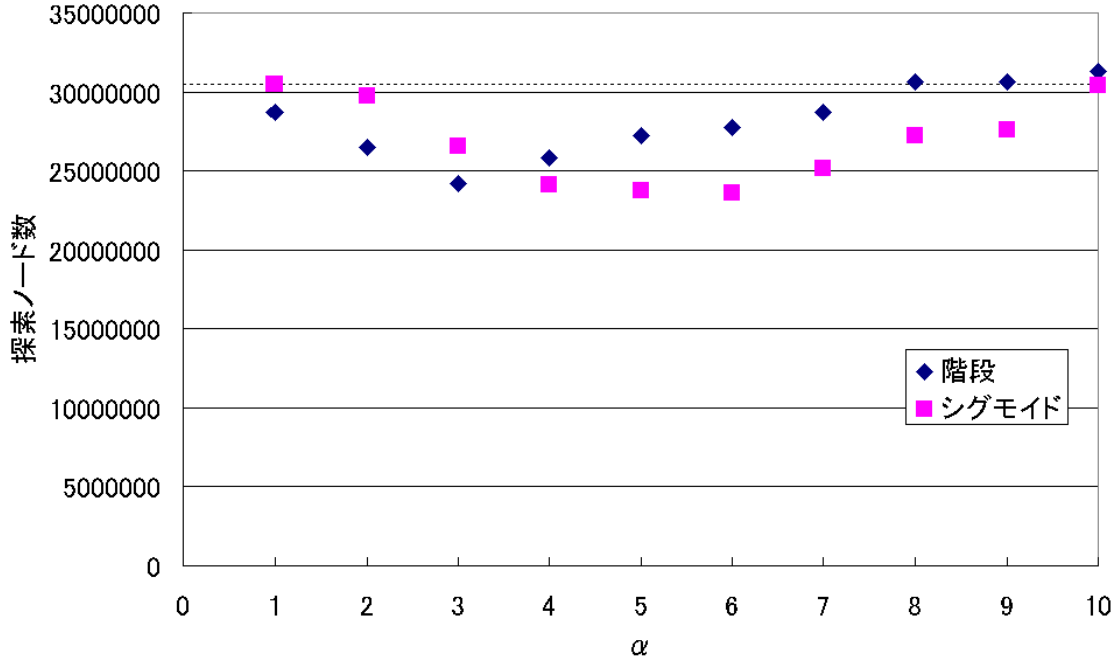

図 4.11: 深さを一定にしてその証明数、反証数の増分を変化させながら探索したときの探索ノード数

表 4.11: 超平面からの距離を考慮しない場合の証明数、反証数の増分

増分＼分類器の評価	詰み	不詰
証明数	$1+\alpha$	1
反証数	1	$1+\alpha$

表 4.12: 超平面からの距離 distance を考慮した場合の証明数、反証数の増分

増分＼分類器の評価	詰み	不詰
証明数	$1+\frac{\alpha}{1+exp(-\mathrm{distance})}$	1
反証数	1	$1+\frac{\alpha}{1+exp(\mathrm{distance})}$

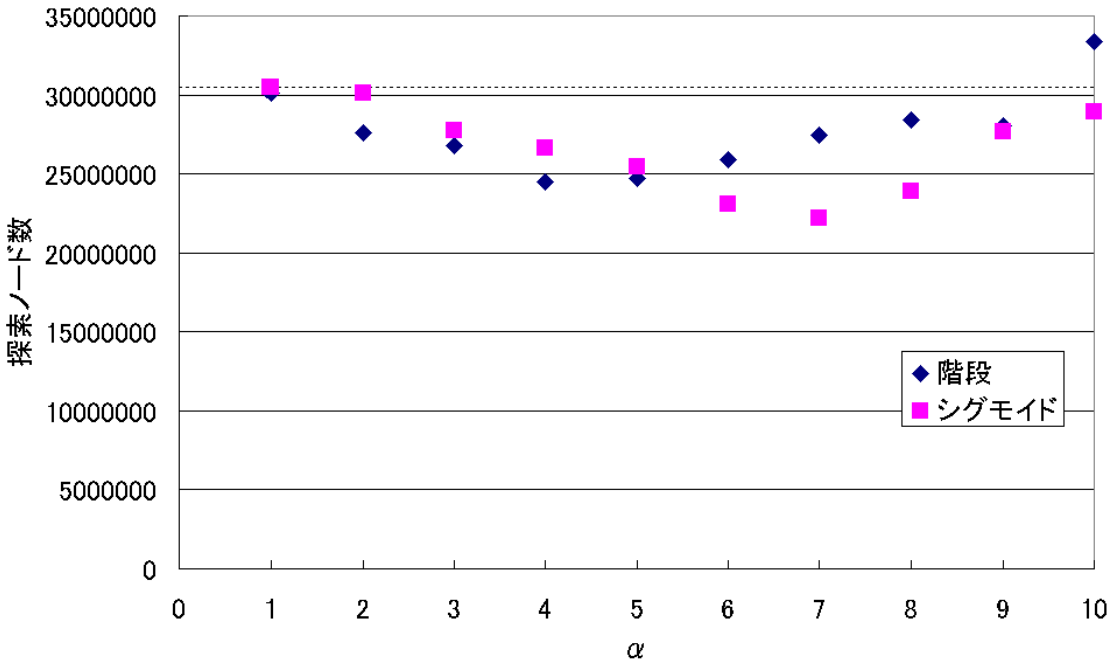

図 4.12: 深さにあわせてその証明数、反証数の増分を減少させながら探索したときの探索ノード数

表 4.13: 深さ depth を考慮し、超平面からの距離を考慮しない場合の証明数、反証数の増分

増分 \ 分類器の評価	詰み	不詰
証明数	1	1+α - depth ($\alpha >$ depth) 1 ($\alpha \leq$ depth)
反証数	1+α - depth ($\alpha >$ depth) 1 ($\alpha \leq$ depth)	1

表 4.14: 深さ depth、超平面からの距離 distance を考慮した場合の証明数、反証数の増分

増分 ＼ 分類器の評価	詰み	不詰
証明数	$1+\frac{\alpha-\text{depth}}{1+exp(-\text{distance})}$ $(\alpha > \text{depth})$ 1 $(\alpha \leq \text{depth})$	1
反証数	1	$1+\frac{\alpha-\text{depth}}{1+exp(\text{distance})}$ $(\alpha > \text{depth})$ 1 $(\alpha \leq \text{depth})$

表 4.15: 証明数、反証数の制御を行った時の探索ノード数の和と探索時間の和

	探索ノード数	探索時間 [sec]
証明数、反証数の制御を行わない場合	3374215066	17529
証明数、反証数の制御を行った場合	2962561231	16048

表 4.16: 探索が不明でない時の証明数、反証数の制御を行った探索ノード数の和と探索時間の和

	探索ノード数	探索時間 [sec]
証明数、反証数の制御を行わない場合	157428878	770
証明数、反証数の制御を行った場合	22451180	119

表 4.17: 証明数、反証数の制御を行った時の解答結果

	詰み	不詰	不明
証明数、反証数の制御を行わない場合	2041	4809	3150
証明数、反証数の制御を行った場合	1913	5366	2721

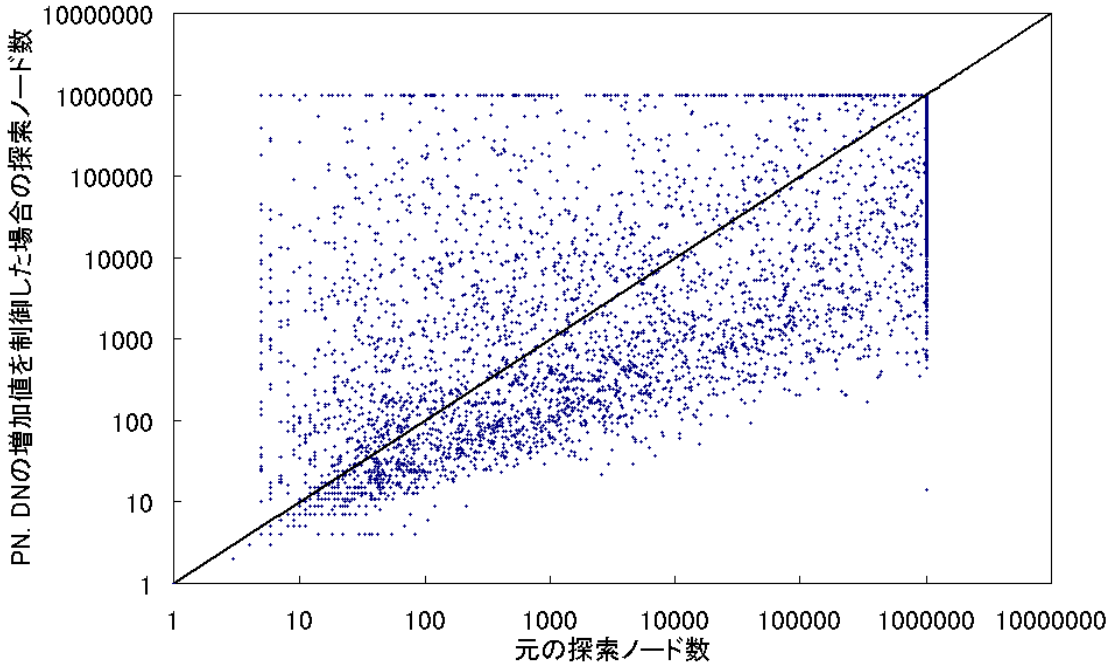

図 4.13: 証明数、反証数の制御を行った時の 1000000 ノード制限で 10000 局面探索したときの探索ノード数の違い

第5章　結論

5.1　まとめ

本研究では、Support Vector Machine を用いた将棋の詰み有無の予測の学習を行った。また、その分類器を用いた応用例として、詰探索 PDS の探索ノード数制限をその詰み予測の結果に応じて変化させること、その詰み予測の結果を PDS の評価関数として用いること、その詰み予測の結果に応じて PDS の証明数、反証数の上限を制御することを試みた。

結果としては、SVM を用いた詰みの有無の学習では 3.3 に示したとおり、86.0%というそれなりの高い正解率を持ち、ほとんどコスト無しに分類可能な分類器を得ることができた。特に探索ノード数が少ないところでの不詰判定器としては非常に高い精度で判別できることもわかった。

また、PDS の探索ノード数制限の制御については、4.3.3 に示したとおり、同等かそれ以上の強さを保ちながら、詰探索が有用な中盤、終盤にかけて、最大で指将棋の探索時間を 61.9%まで削減できることがわかった。

PDS の評価関数として用いることに関しては、4.3.3 に示したとおり、不明でないノードに関しては探索ノード数は 14.6%程度削減されたが、不明のノードも含めると全体として探索時間は 1%程度増加した。

PDS の証明数、反証数の上限の制御については、4.4.3 に示したとおり、不明のノードを含めると 8%程度探索時間が削減され、特に不明でないノードに関しては探索ノード数、探索時間ともに 85%程度削減された。

今回の実験における結果から得られた知見としては大きくは次の 3 つが挙げられる。

- 線形分離を可能にする特徴量の選択の必要性
- 詰みの知識に即した詰探索制御手法の必要性
- 詰みの確率という指標

まず線形分離を可能にする特徴量の選択の必要性について説明する。今回の実験ではカーネルトリックなどを用いず線形の SVM を用いた。この線形でそれなりの分離ができているのは、将棋盤を特徴量に射影する際に線形に分離できる形に射影できているためであると考えられる。高速に分類するためには線形の SVM のような軽い計算ですますことは重要であり、今回用いたような特徴量を将棋盤から抽出することは非常に重要な問題である。また、今回用いたのは SVM であるが、このような軽い計算で分類できる学習器は他にもあり、それらも SVM と同様に有用であろうと考えられる。

次に詰みの知識に即した詰探索制御手法の必要性について説明する。今回用いた手法により将棋全体の探索の探索時間が大量に削減できた。これにより、これまで指将棋の探索において用いられてきたその深さに応じて詰探索の制限を変化させる手法が、必ずしも最善の手法ではないということが分かった。また、この結果よりその制御には詰みの知識に応じてより柔軟に変化させることが有効であるということがわかった。今回の実験で得られたものに限らず、このような詰探索を制御する情報を用いることは有効であると考えられ、これからのコンピュータ将棋をより強いものにするために必須であると考えられる。

最後に詰みの確率という指標について説明する。今回の実験で得られた詰みの確率という指標は、PDS で 5,000,000 ノード探索した際に詰みが見つかるかどうかを学習した結果である。この結果得られた、詰みの確率という指標は、4.3.3、4.4.3 より詰探索の探索ノード数の上限と相関があることがわかった。

5.2 今後の課題

今後の課題としては次のようなことがあげられる。

特徴量の選択・自動抽出 将棋盤から取り出す特徴量の調整は、正解率に直接影響するため非常に重要な要素である。現在は人手で抽出しているが、これには見落としあることが考えられる。また、この人手での抽出にはその特徴量の選択のために多大な時間を要する。このようなことから、単純な特徴量の組み合わせなどにより、これを自動化し、特徴量を抽出することが必要であると考えられえる。

より実際の将棋の傾向に近い学習例による学習 今回はその実験環境の制限などにより、実際の将棋の傾向から外れた学習データを用いた。より計算時間が少ない学習器を用いる、学習を並列化するなどの方法をとることにより、学習データ数を増やして将棋の傾向に近い学習を行ってみることには価値があると考えられる。

他の学習器の利用 今回はその優れた汎化能力から SVM を分類器に選んだが、そのような特徴を持った学習器は他にもあり、SVM と同様に有用であることが考えられる。また、SVM は学習が遅いため、大量のデータを用いて学習することはできなかったが、他の学習器を用いることでそれが可能になることが期待できる。

詰みの確率という指標に関する考察 今回得られた分類器による詰みの確率というのは、PDS で 5,000,000 ノード探索した際に詰みが見つかる確率である。この確率は詰探索の探索ノード数の上限と相関があるということは分かったが、より厳密にはどのような意味を持っているのか、どのように扱われるべきなのかについては考察する必要があると考えられる。この際、実際に詰みの確率というものを求めることは不可能なので、様々な指標と比較する、4.3.3 に書いたような確率として扱ってみるなどの方法をとってみることが考えられる。

詰みの予測の他の応用 今回得られた分類器による詰みの予測は、今回の応用のほかにも例えば候補手の探索順序制御や評価関数に組み込むなどの応用が考えられる。

他のゲームへの応用 詰探索は、将棋に限らず囲碁などにも応用がなされている。今回の手法をそのまま適用するにはその特徴量の抽出について問題があるが、このようなゲームについても本手法のような詰みの予測が可能ではないかと考えられる。

参考文献

[1] Eric B. Baum and Warren D. Smith. A bayesian approach to relevance in game playing. *Artificial Intelligence*, Vol. 97, No. 1-2, pp. 195–242, 1997.

[2] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Learning to play chess using temporal differences. *Machine Learning*, Vol. 40, No. 3, pp. 243–263, 2000.

[3] Bruno Bouzy and Tristan Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, Vol. 132, No. 1, pp. 39–103, 2001.

[4] Chih-Chung Chang and Chih-Jen Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, Vol. 13, No. 9, pp. 2119–2147, 2001.

[5] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines, November 2003. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/` .

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, Vol. 20, No. 3, pp. 273–297, 1995.

[7] R. Grimbergen. Using pattern recognition and selective deepening to solve tsume shogi. In *Game Programming Workshop in Japan '96*, pp. 150–159, Kanagawa, Japan, 1996.

[8] R. Grimbergen. A neural network for evaluating king danger in shogi. 第 7 回 ゲーム・プログラミングワークショップ (GPW2002), pp. 36–43, Kanagawa, Japan, 2002.

[9] R. Grimbergen and H. Matsubara. Pattern recognition for candidate generation in the game of shogi. In *Proceedings of the Workshop on Computer Games (W31) at IJCAI-97*, pp. 7–12, Nagoya, Japan, 1997.

[10] K.-M. Lin and C.-J. Lin. A study on reduced support vector machines. *IEEE Transactions on Neural Networks*, No. 14, pp. 1449–1559, 2003.

[11] D.A. McAllester. Conspiracy Numbers for Min-Max search. *Artificial Intelligence*, Vol. 35, pp. 287–310, 1988.

[12] A Nagai. A new and/or tree search algorithm using proof number and disproof number. In *Proceedings of Complex Games Lab Workshop*, pp. 40–45, Tsukuba, Japan, 1998. ETL.

[13] A. Nagai and H. Imai. Proof for the equivalence between some best-first algorithms and depth-first algorithms for and/or trees. *IEICE Trans. Inform. & Systems*, Vol. E85-D, No. 10, pp. 1645–1653, 2002.

[14] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In B. Schoelkopf D. Schuurmans A.J. Smola, P. Bartlett, editor, *Advances in Large Margin Classiers*, pp. 61–74. MIT Press, 2000.

[15] Johannes Fü rnkranz. Machine learning in games: A survey. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 2, pp. 11–59. Nova Science Publishers, Huntington, NY, 2001.

[16] M. Sakuta and H. Iida. The performance of pn*, pds and pn search on 6x6 othello and tsume-shogi. Technical report, CGRI, 1999.

[17] Daisaku Yokoyama Yoshimasa Tsuruoka and Takashi Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, Vol. 25, No. 3, pp. 145–152, 2002.

[18] 金沢伸一郎. 金沢将棋のアルゴリズム. 松原仁編著（編）, コンピュータ将棋の進歩 3, pp. 15–26. 共立出版, 5 2000.

[19] 薄井克, 鈴木豪, 小谷善行. プロの棋譜を用いたＴＤ法による将棋の評価関数の学習. Technical report, ゲーム情報学, 10 2000.

[20] 垂水清治, 隈智, 山口明宏, 和田充雄. サポートベクタマシンの学習における教師データのフィルタリング手法. 第 10 回インテリジェント・システム・シンポジウム (FAN), 2000.

[21] 三輪誠, 横山大作, 田浦健次朗, 近山隆. Svm を用いた将棋の詰みの有無の予測の学習. 情報処理学会第 66 回全国大会, 3 2004.

[22] 三輪誠, 横山大作, 近山隆. Svm による将棋の詰みの予測とその応用. 第 9 回ゲーム・プログラミング ワークショップ, pp. 143–150, 2004.

[23] 金子知適, 田中哲朗. 将棋プログラムにおける棋譜を利用した囲いの評価. 夏のプログラミングシンポジウム予稿集, 2003.

[24] 金子知適, 田中哲朗, 山口和紀, 川合慧. 駒の関係を利用した将棋の評価関数. 第 8 回ゲーム・プログラミング ワークショップ, pp. 14–21, 2003.

[25] 金子知適, 田中哲朗, 山口和紀, 川合慧. 詰め将棋における df-pn+探索のための、展開後の証明数と反証数を予測する評価関数. 第 9 回ゲーム・プログラミング ワークショップ, pp. 14–21, 2004.

[26] 川田敦史. 将棋の局面情報の学習に関する研究. Master's thesis, 名古屋工業大学, 2001.

[27] 有岡雅章. 将棋プログラム kfend における探索. 松原仁編著（編）, アマ 4 段を超える コンピュータ将棋の進歩 4, pp. 18–40. 共立出版, 7 2003.

発表文献

[1] 三輪誠, 横山大作, 田浦健次朗, 近山隆. Svm を用いた将棋の詰みの有無の予測の学習. 情報処理学会第 66 回全国大会, 3 2004.

[2] 三輪誠, 横山大作, 近山隆. Svm による将棋の詰みの予測とその応用. 第 9 回ゲーム・プログラミング ワークショップ, pp. 143–150, 2004.

謝辞

本研究を進めるにあたり、多くの方にお世話になりました。

近山隆教授、田浦健次朗助教授には、論文作成やプレゼンテーションの方法など多くのご指導、ご教示を頂きました。

横山大作助手には、研究生活から研究内容、その進め方まで多岐にわたって日頃から相談に乗って頂きました。

他の近山・田浦研究室の皆様にも公私にわたり多くの助言を頂きました。

ここに、心より感謝の意を表します。

平成 17 年 1 月 31 日