4E−9　バイアスの変更に対応できる学習戦略

山田太一　犬塚信博　石井直宏

名古屋工業大学

1．はじめに

帰納的学習問題を定義する場合、ユーザは扱う対象空間、推論方法、概念の表現などを決める必要がある。そして、これらを決めることによりその学習結果には偏りが生じる（得られる結果の枠がきまる）。この偏りをバイアスという。正しい結果を得るためにユーザはその問題にふさわしいバイアスを決める必要がある。従来の方法では、このバイアスは学習前に決められ、学習途中でそれが変更されることはない。しかし実際には、学習前に決められたバイアスが最も適切であるかは保証されていないことが多い。そこで本研究では、学習戦略として有用なバージョン空間法[1]をバイアスの変更に対応できるように拡張した。

2．バージョン空間法

バージョン空間法とは、次々と与えられる実例に無矛盾なすべての仮説の集合Hを、仮説空間内で求めながら学習を進めてゆく戦略である。但し、この戦略ではHをその中で極大に一般的な仮説の集合Gと極大に特殊な仮説の集合Sによって挟まれた領域（バージョン空間）として表現する。

そして、正の実例が入力されるとH内の仮説がそれを含むようにS内の仮説を一般化し、負の実例が入力されるとH内の仮説がそれを排除するようにG内の仮説を特殊化する。これらの処理によってHの中の仮説が唯一つとなったとき、学習は終了する。そして、それが求めるべき概念である。

3　扱う学習問題とバイアスの変更

ここで扱う学習問題は次のような実例記述言語で表現された実例が与えられたとき、それを説明する概念を概念記述言語で記述する問題である。

(1)実例記述言語

属性値の連言表現で実例を記述する。例えば、赤い三角形という実例を「三角∧赤」で記述する。

(2)概念記述言語

属性値の連言表現で概念を記述する。但し、属性値として実例の集合に対応する言葉も使用できる。例えば、赤い多角形という概念を「多角形∧赤」で記述する。ここで、各属性の属性値間には一般−特殊という半順序関係が成り立つ。この関係を図1のようなグラフで表現し、これを属性グラフと呼ぶ。但し、a→bのときaはbより一般的（bはaより特殊）である。

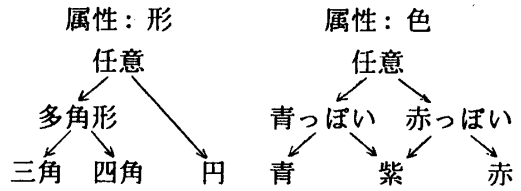

図1　属性グラフ

そして、ここで対象とするバイアスの変更は、図2のような属性の変更に等価である。つまり、ある属性において、いくつかの属性値を包含する新しい属性値を加えるという変更である。

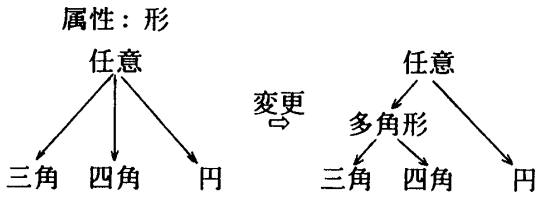

図2　属性の変更

4　拡張されたバージョン空間法

拡張されたバージョン空間法のアルゴリズムを図3に示す。本方法では、属性の変更に対応するために次のようなG_markとS_markをバージョン空間法のGとSの代わりに計算する。

G_mark

G_markはNode_Patternの集合からなる。Node_Pattern は与えられた負の実例に対する見方を示している。例えば、図1のような属性の下で、二つの負の実例「三角∧赤」,「円∧青」が与えられた

Learning strategy for modifying conceptual bias
Taichi YAMADA, Nobuhiro INUZUKA, Naohiro ISHII
Nagoya Institute of Technology

```
main;                    i      m
  1.G_mark= { ( φ,....,φ) } ; G_markの初期化
  2.S_mark= ( φ,....,φ)     ; S_markの初期化
  3.while not(End)
  4.     実例の入力              ⇒  (a)
  5.     属性の変更              ⇒  (b)
  6.     バージョン空間の表示    ⇒  (c)
  7.end.
```

(a)実例の入力

```
 1. i =read()
 2.if i が負の実例である then
 3.  G_mark′ = φ
 4.  for (M1,..,Mj,...,Mm)∈G_mark do
 5.   for j=1 to m do
 6.    G_mark′=G_mark′∪
        {(M1,...,Mj∪{ij の先祖},...,Mm)}
 7.   fend
 8.  fend
 9.  G_mark=G_mark′
10.else ; i が正の実例である。
11.  S_mark′= (P1∪{i1},...,Pm∪{im})
     where S_mark= (P1,....,Pm)
12.  S_mark=S_mark′
13.end.
```

(b) 属性の変更

```
 1.j=属性の識別番号
 2.xnew=新しく加える属性値
 3.属性Xjにxnewを加える。
 4.G_mark′ = φ
 5.for  (M1,...,Mj,...,Mm)∈G_mark do
 6.if   (Mj ∩ {xnewの子供}) ≠ φ
 7.   then G_mark′=G_mark′∪
            {(M1,...,Mj∪{xnew},...,Mm)}
 8.   else G_mark′=G_mark′∪
            {(M1,...,Mj,...,Mm)}
 9.fend
10.G_mark=G_mark′
11.end.
```

(c)バージョン空間の表示

```
1. G′={G_markの各Node_Patternから求めた負
       の実例を排除する概念の中で最も一般
       的な概念}
2. S′={S_markから求めた正の実例を包含する
       概念の中で最も特殊な概念}
3. G={S′のある概念より一般的なG′の概念}
4. S={G′のある概念より特殊なS′の概念}
```

図3　アルゴリズム

とする。ここで、実例は属性値の連言で記述されているため、負の実例の少なくとも一つの属性値が目標概念に矛盾するはずである。この場合、例えば「三角」と「青」が矛盾するという一つの見方ができる。そしてこれをその属性値の先祖を書き並べることで次のように表し、Node_Paternと呼ぶ。

((三角,多角形,任意),(青,青っぽい,任意))

このNode_Patternの持つ属性値以外の属性値を使って概念を作れば（例えば、「円∧赤っぽい」等）、与えられた負の実例を排除する概念を得ることができる。また、属性の変更を行なったときには、新しい属性値の子供を持つNode_Patternに対しその先祖としてその属性値を加える（図3(b)）。

S_mark

S_markは正の実例として現われた属性値を各属性に対して記憶したものである。例えば、図1の属性の下で「四角∧紫」、「円∧赤」が入力されたとするとこのときS_markは次のようになる。

((四角,円),(紫,赤))

ここで、実例を属性値の連言で記述しているため、正の実例として現われた属性値は目標概念に必ず包含される。従って、S_markが持つ属性値を全て包含する属性値から概念を作れば、与えられた正の実例を包含する概念を得ることができる。また、S_markは各属性グラフの葉（属性値）からなるので、属性の変更には無関係である。

5. 情報の削減

このアルゴリズムではG_markが保持するNode_Pattern の個数が属性数と各属性グラフのノード数に対して指数的に大きくなり、計算時間に影響を与える。そこで、記憶するNode_Patternの数を減らすために次の二つの削除方法を行なっている。

(1) G_markの中のNode_Patternはできる限り属性値の数が少ないもののみにする。

(2) G_markの中でS_markのある属性値を持つNode_Patternを削除する。

6. まとめ

本研究では、バージョン空間法をバイアスの変更に対応できるように拡張した。またその正当性を示すこともできる。[2] 従来のバージョン空間法ではこの変更に対応するには再計算の必要があるのに対しこの方法ではその必要がない。しかし、この方法では直接バージョン空間を保持していないためそれを表示するためにはG_markとS_markからGとSを求める必要があり、表示するかしないかによって計算時間に差が生じる。また、5. で示した方法で削除を行なってもNode_Patternの個数が多くなる場合があるため、この情報の持ち方を工夫する必要がある。

参考文献

1) Mitchell,T.M., "Generalization as Search", Artif.Intell.,18,pp.203-226,1982 .

2) 山田,犬塚,石井, "概念記述言語の更に対応できる学習戦略",信学技報,AI90-69,1990.