Projet d'IA

Classification des champignons

Sébastien Derivaux
esotech@free.fr

12 avril 2004

Table des matières

Introduction

L'objectif du sujet était de créer un système expert permettant d'aider à la classification de champignons. Une description de 28 champignons extraite du livre de Pierre MONTAMAL, *Le petit livre des champignons*, servait de source de connaissance. Il a fallut en extraire les faits décrivant les champignons et les règles définissant quels faits permettaient d'identifier un champignon particulier. J'ai implémenté ce système expert en utilisant un moteur d'inférence par chaînage arrière programmé en C++.

1 La base de faits

1.1 Format des faits

Dans notre système expert, chaque fait est de la forme *type*(*valeur*). Le système utilise donc la logique d'ordre 0+, celle des prédicats. Un fait est ainsi défini afin de réduire les interaction avec l'utilisateur. En effet, on a, par exemple, le type *LamellesExt* qui peut avoir plusieurs valeurs rarement plus d'une à la fois, plutôt que de demander si les lamelles ont des plis épais ou pas, si elles sont décurrentes ... on ne demande qu'une fois ce qui les décrit (l'utilisateur peut rentrer plusieurs valeurs en les séparant d'un espace. Cela nous permet aussi de gérer facilement le cas où l'utilisateur ne sait pas, il rentre *nonconnue* par exemple pour *LamellesExt* ce qui donnera *LamellesExt*(*nonconnue*), ainsi le système sera informé que l'utilisateur a déjà répondu à la question et donc ne lui reposera pas.

1.2 Descriptions des types de faits

Les champignons ont été décrit selon 8 types de fait décrit ci-dessous.

Chair : Décrit la consitance de la chaire du champignon, peut valoir *grenue* si la chaire est grenue et cassante (ces deux attribut allant de paire).

Lait : *oui* si le champignon a du lait, *non* s'il n'en a pas.

Lamelles : *oui* si le champignon a des lamelles.

LCouleur : Définit la couleur des lamelles. Les valeurs possibles sont : *gris* et *rose-violet* (signifiant rose ou violet). L'utilisateur pouvant n'avoir qu'un seul champignon d'un genre peut utiliser *rose* et *violet* qui donnerons *rose-violet* (aucun genre de champignons n'est que rose ou que violet).

LamellesExt : Définit des aspects des lamelles du chapignon, ce type peut prendre les valeurs : *plisepais* pour des lamelles à plis épais, *decur* pour des lamelles décurrentes, *deliq* pour des lamelles déliquescentes, *echan* si les lamelles sont échancrées, *adher* si elles sont adhérentes, *libre* si elles sont libres. Pour les règles il y a aussi les combinaisons de ces attributs.

PiedDetach : *oui* si le pied peut se détacher facilement, *non* si cela implique la déchirure de la chaire.

PiedPos : *central* ou *excentré* selon la position du pied.

PiedType : Un pied peut être *cartilagineux* ou *tubuleux*.

Spore : La couelurs de spore si le champignons en a : *blanc*, *rose*, *ocre* ou *noir*.

ACV : Décrit des composantes que peut avoir le champignons : une *volve*, un *anneau* et/ou une *cortine*.

1.3 Tableau des attributs

Champignon	Chair	Lait	Lamelles	LCouleur	LamellesExt	PiedDetach	PiedPos	PiedType	Spore	ACV
Amanita	?	?	oui	?	?	?	?	?	blanc	volve
Armillaria	?	?	oui	?	?	non	?	?	?	anneau
Cantharellus	?	?	oui	?	plisepais	?	?	?	blanc	?
Clitocybe	?	?	oui	?	decur	?	central	?	blanc	?
Clitopilus	?	?	oui	?	?	?	?	?	rose	?
Collybia	?	?	oui	?	?	?	?	cartilagineux	blanc	?
Coprinus	?	?	oui	?	deliq	?	?	?	?	?
Cortinarius	?	?	oui	?	?	?	?	?	ocre	cortine
Entoloma	?	?	oui	?	echan	?	?	?	rose	?
Gomphidius	?	?	oui	gris	decur	?	?	?	noir	?
Hebeloma	?	?	oui	?	echan	?	?	?	ocre	?
Hygrophorus	?	?	oui	?	ecart-epais-decur	?	?	?	blanc	?
Hypholoma	?	?	?	?	?	?	?	?	noir	cortine
Inocybe	?	?	oui	?	adher	?	?	?	ocre	?
Laccaria	?	?	oui	rose-violet	ecart-decur	?	?	?	?	?
Lactarius	grenue	oui	?	?	?	?	?	?	?	?
Lepiota	?	?	oui	?	libre	oui	?	?	?	anneau
Marasmius	?	?	?	?	?	?	?	?	blanc	?
Mycena	?	?	oui	?	?	?	?	tubuleux	?	?
Paxillus	?	?	oui	?	decur	oui	?	?	ocre	?
Pholiota	?	?	oui	?	?	?	?	?	ocre	anneau
Pleurotus	?	?	oui	?	decur	?	excentré	?	blanc	?
Pluteus	?	?	oui	?	libre	?	?	?	rose	?
Psalliota	?	?	?	?	?	?	?	?	noir	?
Russula	grenue	non	?	?	?	?	?	?	?	?
Stropharia	?	?	?	?	?	?	?	?	noir	anneau
Tricholoma	?	?	oui	?	echan	?	?	?	blanc	?
Volvaria	?	?	?	?	?	?	?	?	rose	volve

1.4 Pertinence des faits

Pour tester si les faits ainsi dégagés étaient suffisant pour classer les champignons, j'ai utilisé l'algorithme C45 implémenté dans Weka[1] sous le nom de J48. Cet algorithme génère un arbre de classification avec les données fournies, les valeurs inconnues ont été nommé *na*. Comme on peut le voir, chaque feuille n'a qu'un seul seul champignon, qui donne son nom à la feuille.

```
Lamelles = oui
|   LamellesExt = plisepais: Cantharellus (1.0)
|   LamellesExt = decur
|   |   PiedPos = central: Clitocybe (1.0)
|   |   PiedPos = excentré: Pleurotus (1.0)
|   |   PiedPos = na
|   |   |   LCouleur = gris: Gomphidius (1.0)
|   |   |   LCouleur = na: Paxillus (1.0)
|   LamellesExt = deliq: Coprinus (1.0)
|   LamellesExt = echan
|   |   Spore = blanc: Tricholoma (1.0)
|   |   Spore = rose: Entoloma (1.0)
|   |   Spore = ocre: Hebeloma (1.0)
|   LamellesExt = ecart-epais-decur: Hygrophorus (1.0)
|   LamellesExt = adher: Inocybe (1.0)
```

[1] logiciel de fouille de données disponible sur http ://www.cs.waikato.ac.nz/∼ml/weka/

```
|   LamellesExt = ecart-decur: Laccaria (1.0)
|   LamellesExt = libre
|   |   PiedDetach = oui: Lepiota (1.0)
|   |   PiedDetach = na: Pluteus (1.0)
|   LamellesExt = na
|   |   Spore = blanc
|   |   |   PiedType = cartilagineux: Collybia (1.0)
|   |   |   PiedType = na: Amanita (1.0)
|   |   Spore = rose: Clitopilus (1.0)
|   |   Spore = ocre
|   |   |   ACV = anneau: Pholiota (1.0)
|   |   |   ACV = cortine: Cortinarius (1.0)
|   |   Spore = na
|   |   |   PiedDetach = non: Armillaria (1.0)
|   |   |   PiedDetach = na: Mycena (1.0)
Lamelles = na
|   Spore = blanc: Marasmius (1.0)
|   Spore = rose: Volvaria (1.0)
|   Spore = noir
|   |   ACV = anneau: Stropharia (1.0)
|   |   ACV = cortine: Hypholoma (1.0)
|   |   ACV = na: Psalliota (1.0)
|   Spore = na
|   |   Lait = oui: Lactarius (1.0)
|   |   Lait = non: Russula (1.0)
```

Cela dit, si on a unicité du classement de chaque champignons, cela n'est vrai que si pour chaque trait non renseigné par notre description de champignons, l'utilisateur n'est pas en mesure d'évaluer ce trait.

Si nous prennons par exemple le Russula et le Stropharia. Si l'utilisateur est en face d'un champignons à chair grenue, sans lait avec des spore noires et un anneau, nous n'avons aucun moyen de savoir s'il s'agit d'un Russula ou d'un Stropharia. Ce problème est du au fait qu'en général seul les traits existant sont spécifié, mais nous ne pouvons pas savoir si un champignon a, ou n'a pas, un anneau si la description ne le dit pas.

Cela a pour effet d'entrainer que lors d'une utilisation du système expert créé, celui-ci retourne souvent plusieurs genre possible de champignons.

2 La base de règles

2.1 Règles de classification des champignons

Les règles de classification des champignons sont extraite du tableau des attributs. Elle peuvent être trouvé dans l'appendice B. Voici un exemple sous une notation Prolog :

$$Champignon(Amanita) :- Lamelles(oui), Spore(blanc), ACV(volve).$$

Cela signifie que pour avoir le fait $Champignon(Amanita)$, il faut que les fait $Lamelles(oui)$, $Spore(blanc)$ et $ACV(volve)$ existent.

2.2 Règles secondaires

Dans la description des types de faits qui a été faite précédemment, nous avons vu que certains types pouvaient avoir des valeurs composée comme $rose-$

violet pour $LCouleur$ et $ecart - decur$ pour $LamellesExt$. Ces valeurs composées demandent de rajouter des règles pour les prendre correctement en compte.

Nous avons procédé ainsi pour que chaque type n'ait qu'une valeur ce qui rend la saisie beacoup plus simple. Pour le cas de $LamellesExt$ les valeurs composées correspondent à la présence de tout les attributs mis dans la composition, la règle est donc :

$$LamellesExt(X - Y) : -LamellesExt(X), LamellesExt(Y).$$

Avec X et Y des valeurs possible de $LamellesExt$ et $X - Y$ la composition de ces deux valeurs.

Pour $LCouleur$ le problème est différent, nous avons $rose - violet$ car certains champignons sont roses et d'autre violets mais appartenant au même genre. C'est donc un OU logique qui est effectué.

$$LCouleur(X - Y) : -LCouleur(X).$$
$$LCouleur(X - Y) : -LCouleur(Y).$$

Enfin, des règles ont été ajoutée pour éviter de poser des question auxquelles le système expert a déjà les réponses. C'est le cas pour $Lamelle$, inutile de poser la question de l' existance de lamelles si on a déjà des informations sur ces lamelles.

Notre moteur d'inférence ne travaillant pas en logique d'ordre 1, il a fallu écrire une règle pour chaque valeur possible. Elle peuvent être retrouvée dans le fichier de règles à l'appendice B.

3 Le moteur d'inférence

3.1 Chaînage arrière

Le moteur d'inférence du système expert mis en oevre fonctionne par chaînage arrière. Il commence à partir de l'endroit où il veut arriver, dans notre cas trouver des faits de la forme $Champignon(X)$, il va donc prendre chaque règle et regarde si son objectif est de générer un tel fait. Si c'est le cas, alors le système regarde les prémisses de cette règle et essaye de les vérifier ; soit elles existaient déjà dans notre base de fait, soit on peut les inférer avec une autre règle, soit on demande la valeur du fait à l'utilisateur.

3.2 Exemple

```
Quelle est la valeur de LamellesExt ?
decur epais ecart
Quelle est la valeur de Spore ?
blanc
Quelle est la valeur de ACV ?
volve
Quelle est la valeur de PiedDetach ?
oui
Quelle est la valeur de PiedPos ?
centre
```

```
Quelle est la valeur de PiedType ?
chaispas
Quelle est la valeur de LCouleur ?
rose
Quelle est la valeur de Chair ?
chaispas

==> Résultats trouvés:
Amanita
Hygrophorus
Marasmius

==> Etat de la base de faits après la recherche:
LamellesExt(decur).
LamellesExt(epais).
LamellesExt(ecart).
Lamelles(oui).
Lamelles(oui).
Spore(blanc).
ACV(volve).
Champignon(Amanita).
PiedDetach(oui).
PiedPos(centre).
PiedType(chaispas).
LCouleur(rose).
LamellesExt(ecart-epais-decur).
LamellesExt(ecart-epais-decur).
Champignon(Hygrophorus).
Chair(chaispas).
Champignon(Marasmius).
```

Dans cet exemple le système fonctionne de la manière suivante ; il recherche à trouver des faits de la forme $Champignon(X)$, il regarde la première règle de fichier de règles :

$$Champignon(Cantharellus) :- \\ LamellesExt(plisepais), Lamelles(oui), Spore(blanc).$$

Cette règle a été placée en première dans notre fichier de règle car elle contient $LamellesExt$ ce qui fera que la question LamellesExt sera posée avant Lamelles et donc si on réponds une bonne valeur à LamellesExt, le système infèrera tout seul $Lamelles(oui)$.

Le système n'ayant aucun fait pour l'intant et aucune règle ne permettant de déduire $LamellesExt(plisepais)$, le système va demander à l'utilisateur qui répond qu'il existe les faits $LamellesExt(decur)$, $LamellesExt(epais)$ et $LamellesExt(ecart)$, comme il n'y a pas $LamellesExt(plisepais)$, le système laisse tomber cette règle est passe à la suivante qui est :

$$Champignon(Amanita) :- Lamelles(oui), Spore(blanc), ACV(volve).$$

Le système trouve une règle qui permet d'inférer $Lamelles(oui)$ à partir de $LamellesExt(ecart)$ que nous avons dans notre base de fait. Pour $Spore(blanc)$ comme il n'existe pas de règles il demande à l'utilisateur qui ici valide ce fait.

Enfin de même pour $ACV(volve)$. Le système génère donc le fait $Champignon(Amanita)$.

Les règles suivantes ne sont pas satisfaites, jusqu'à arriver à la règle :

$$Champignon(Hygrophorus) :- \\ LamellesExt(ecart - epais - decur), Lamelles(oui), Spore(blanc).$$

Là il infère $LamellesExt(ecart - epais - decur)$ à partir de la règle :

$$LamellesExt(ecart - epais - decur) :- \\ LamellesExt(ecart), LamellesExt(epais), LamellesExt(decur).$$

Et donc génère le fait $Champignon(Hygrophorus)$. Par là suite, seul le champignon Marasmius sera encore ajouté à notre base des faits. Le système a donc trouvé 3 champignons qui sont décrit par les informations données par l'utilisateur.

Conclusion

Ce sujet m'a permit d'étudier par la pratique le fonctionnement des systèmes experts et les difficultées qui sont rencontrées lors de leurs créations. L'aspect le plus sensible est de convertir une information littéraire, très peu formelle, en un ensemble de faits et de règles. Néanmoins il reste qu'un système expert est simple à mettre en oeuvre et relativement performant dans le domaine d'aide à la déçision.

A Programme

Listing 1 – Champignons.cpp

```
1  #include <string>
2  #include <iostream>
3  #include <fstream>
4  #include <sstream>
5  #include <list>
6
7  using namespace std;
8
9  void InfereValeur(string type, string valeur);
10 void DemandeValeur(string goal);
11
12 struct Fait
13 {
14     string mType;
15     string mValeur;
16
17     // Fake Constructeur
18     Fait() {}
19
20     Fait(string type, string valeur)
21     { mType = type; mValeur = valeur; }
22
23     operator string()
24     {
25         string str(mType + "(" + mValeur + ")");
```

```
26          return str;
27      }
28  };
29
30  struct Regle
31  {
32      list<Fait>  mIf;
33      Fait        mThen;
34
35      Regle(Fait th) { mThen = th; }
36
37      Regle& operator<<(Fait fait) { mIf.push_back(fait); return *
            this; }
38
39      operator string()
40      {
41          stringstream str;
42          for(list<Fait>::iterator it = mIf.begin(); it != mIf.end();
                ++it)
43              str << (string)(*it) << " ";
44
45          str << "=> " << (string)(mThen);
46          return str.str();
47      }
48  };
49
50  list<Fait>  gFaits;
51  list<Regle> gRegles;
52
53  void Lire(string fichier)
54  {
55      ifstream input(fichier.c_str());
56
57      string type;
58      string valeur;
59
60      while(input.good())
61      {
62          stringstream str;
63          input.get(*str.rdbuf());
64          str >> type;
65          str >> valeur;
66
67          Regle r(Fait(type, valeur));
68
69          while (!str.eof())
70          {
71              str >> type;
72              str >> valeur;
73              r << Fait(type, valeur);
74          }
75
76          gRegles.push_back(r);
77
78          input.get(); // On lit le \n
79      }
80
81      input.close();
82  }
83
84  // Précondition IsExist(type)
85  Fait& ChercheFait(string type)
```

```
86  {
87      for(list<Fait>::iterator it = gFaits.begin(); it != gFaits.end
            (); ++it)
88      {
89          Fait& f=*it;
90          if(f.mType == type)
91              return f;
92      }
93      throw 1;
94  }
95
96  // Cherche si un type de fait est dans notre base de faits.
97  bool IsExist(string type)
98  {
99      for(list<Fait>::iterator it = gFaits.begin(); it != gFaits.end
            (); ++it)
100     {
101         Fait& f=*it;
102         if(f.mType == type)
103             return true;
104     }
105     return false;
106 }
107
108 // Cherche si un fait est dans notre base de faits.
109 bool IsTrue(string type, string valeur)
110 {
111     for(list<Fait>::iterator it = gFaits.begin(); it != gFaits.end
            (); ++it)
112     {
113         Fait& f=*it;
114         if(f.mType == type && f.mValeur == valeur)
115             return true;
116     }
117     return false;
118 }
119
120 // Rajoute un fait dans la base de faits
121 void Assert(string type, string valeur)
122 {
123     gFaits.push_back(Fait(type, valeur));
124 }
125
126 // Applique une règle.
127 // Renvoie true si on peut appliquer la règle.
128 bool AppliqueRegle(Regle& r)
129 {
130     static Regle* gDerniereRegle=0;
131
132     for(list<Fait>::iterator it = r.mIf.begin(); it != r.mIf.end();
            ++it)
133     {
134         Fait& f=*it;
135         // On regarde si on peut déduire la valeur.
136         InfereValeur(f.mType, f.mValeur);
137         // On ne demande pas si on a déjà une valeur.
138         if(!IsExist(f.mType))
139             DemandeValeur(f.mType);
140         if(!IsTrue(f.mType, f.mValeur))
141             return false;
142     }
143
```

```
144     // Toutes les prémisses sont validé on ajoute le fait.
145     Assert(r.mThen.mType, r.mThen.mValeur);
146 
147     return true;
148 }
149 
150 void InfereValeur(string type, string valeur)
151 {
152     // On tente de générer le fait avec des règles.
153     list<Regle>::iterator it = gRegles.begin();
154     while(it != gRegles.end())
155     {
156         Regle& r = *it;
157         // Si la règle déduit notre but on cherche si les
                conditions sont bonnes.
158         if(r.mThen.mType == type && r.mThen.mValeur == valeur)
159         {
160             if(AppliqueRegle(r))
161             {
162                 list<Regle>::iterator old = it;
163                 ++it;
164                 gRegles.erase(old); // On retire la règle
165             }
166             else
167                 ++it;
168         }
169         else
170             ++it;
171     }
172 }
173 
174 void DemandeValeur(string goal)
175 {
176     // On demande à l'utilisateur.
177     string reponse;
178     cout << "Quelle est la valeur de " << goal << " ?" << endl;
179     // Chaque réponse est séparée par un espace.
180     stringstream str;
181     cin.get(*str.rdbuf());
182     while(str.good())
183     {
184         str >> reponse;
185         Assert(goal, reponse);
186     }
187     // On mange le \n.
188     cin.ignore();
189 }
190 
191 void Recherche(string goal)
192 {
193     list<string> resultats;
194 
195     // On tente de générer le fait avec des règles.
196     list<Regle>::iterator it = gRegles.begin();
197     while(it != gRegles.end())
198     {
199         Regle& r = *it;
200         // Si la règle déduit notre but on cherche si les
                conditions sont bonnes.
201         if(r.mThen.mType == goal)
202         {
203             if(AppliqueRegle(r))
```

```
204                 {
205                     resultats.push_back(r.mThen.mValeur);
206                     list<Regle>::iterator old = it;
207                     ++it;
208                     gRegles.erase(old);
209                 }
210                 else
211                     ++it;
212             }
213             else
214                 ++it;
215         }
216
217         // On affiche les résultats
218         if(resultats.size())
219         {
220             cout << "\n==> Résultat trouvé: " << endl;
221             for(list<string>::iterator it = resultats.begin();
222                 it != resultats.end(); ++it)
223                 cout << *it << endl;
224         }
225         else
226             cout << "Pas de résultats trouvé à votre recherche." <<
                endl;
227
228         cout << "Press enter." << endl;
229         cin.get();
230
231         // On affiche la base de faits.
232         cout << "\n==> Etat de la base de faits après la recherche: "
            << endl;
233         for(list<Fait>::iterator it = gFaits.begin(); it != gFaits.end
            (); ++it)
234         {
235             Fait& f=*it;
236             cout << f.mType << "(" << f.mValeur << ")." << endl;
237         }
238
239     }
240
241     int main()
242     {
243         // On lit les règles.
244         Lire("regles.txt");
245
246         // On cherche les candidats champignons.
247         Recherche("Champignon");
248
249         cout << "Press enter." << endl;
250         cin.get();
251     }
```

B Fichier de règles

Les règles sont construite ainsi :

```
règle ::= conséquence [condition]*
conséquence ::= type valeur
condition ::= type valeur
```

Listing 2 – regles.txt

```
1  Champignon Cantharellus LamellesExt plisepais Lamelles oui Spore
       blanc
2  Champignon Amanita Lamelles oui Spore blanc ACV volve
3  Champignon Armillaria Lamelles oui PiedDetach non ACV anneau
4  Champignon Clitocybe LamellesExt decur Lamelles oui PiedPos central
        Spore blanc
5  Champignon Clitopilus Lamelles oui Spore rose
6  Champignon Collybia Lamelles oui PiedType cartilagineux Spore blanc
7  Champignon Coprinus LamellesExt deliq Lamelles oui
8  Champignon Cortinarius Lamelles oui Spore ocre ACV cortine
9  Champignon Entoloma LamellesExt echan Lamelles oui Spore rose
10 Champignon Gomphidius LamellesExt decur Lamelles oui LCouleur gris
       Spore noir
11 Champignon Hebeloma LamellesExt echan Lamelles oui Spore ocre
12 Champignon Hygrophorus LamellesExt ecart-epais-decur Lamelles oui
       Spore blanc
13 Champignon Hypholoma Spore noir ACV cortine
14 Champignon Inocybe LamellesExt adher Lamelles oui LCouleur rose-
       violet Spore ocre
15 Champignon Laccaria LamellesExt ecart-decur Lamelles oui
16 Champignon Lactarius Chair grenue Lait oui
17 Champignon Lepiota LamellesExt libre Lamelles oui PiedDetach oui
       ACV anneau
18 Champignon Marasmius Spore blanc
19 Champignon Mycena Lamelles oui PiedType tubuleux
20 Champignon Paxillus LamellesExt decur Lamelles oui PiedDetach oui
       Spore ocre
21 Champignon Pholiota Lamelles oui Spore ocre ACV anneau
22 Champignon Pleurotus LamellesExt decur Lamelles oui PiedPos
       excentré Spore blanc
23 Champignon Pluteus LamellesExt libre Lamelles oui Spore rose
24 Champignon Psalliota Spore noir
25 Champignon Russula Chair grenue Lait non
26 Champignon Stropharia Spore noir ACV anneau
27 Champignon Tricholoma LamellesExt echan Lamelles oui Spore blanc
28 Champignon Volvaria Spore rose ACV volve
29 LCouleur rose-violet LCouleur rose
30 LCouleur rose-violet LCouleur violet
31 LamellesExt ecart-epais-decur LamellesExt ecart LamellesExt epais
       LamellesExt decur
32 LamellesExt ecart-epais-decur LamellesExt ecart LamellesExt decur
33 Lamelles oui LamellesExt ecart
34 Lamelles oui LamellesExt plisepais
35 Lamelles oui LamellesExt decur
36 Lamelles oui LamellesExt echan
37 Lamelles oui LamellesExt adher
38 Lamelles oui LamellesExt libre
39 Lamelles oui LamellesExt deliq
```