Projet Foot
Cours 5
2I013

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr

Université Pierre et Marie Curie (UPMC)
Laboratoire d'Informatique de Paris 6 (LIP6)

S2 (2015-2016)

Plan

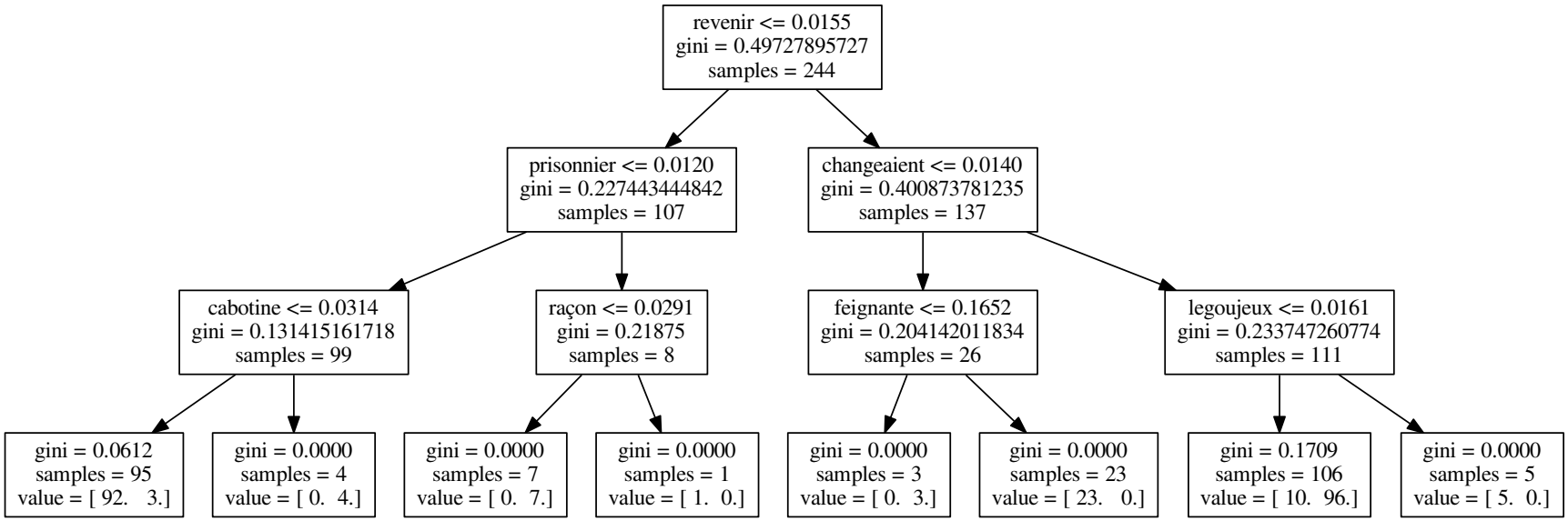
Tournoi 1v1

Tournoi 2v2

Tournoi 4v4

Arbres de décision (rappel)

Objectif : Apprentissage pour le choix automatique d'une stratégie

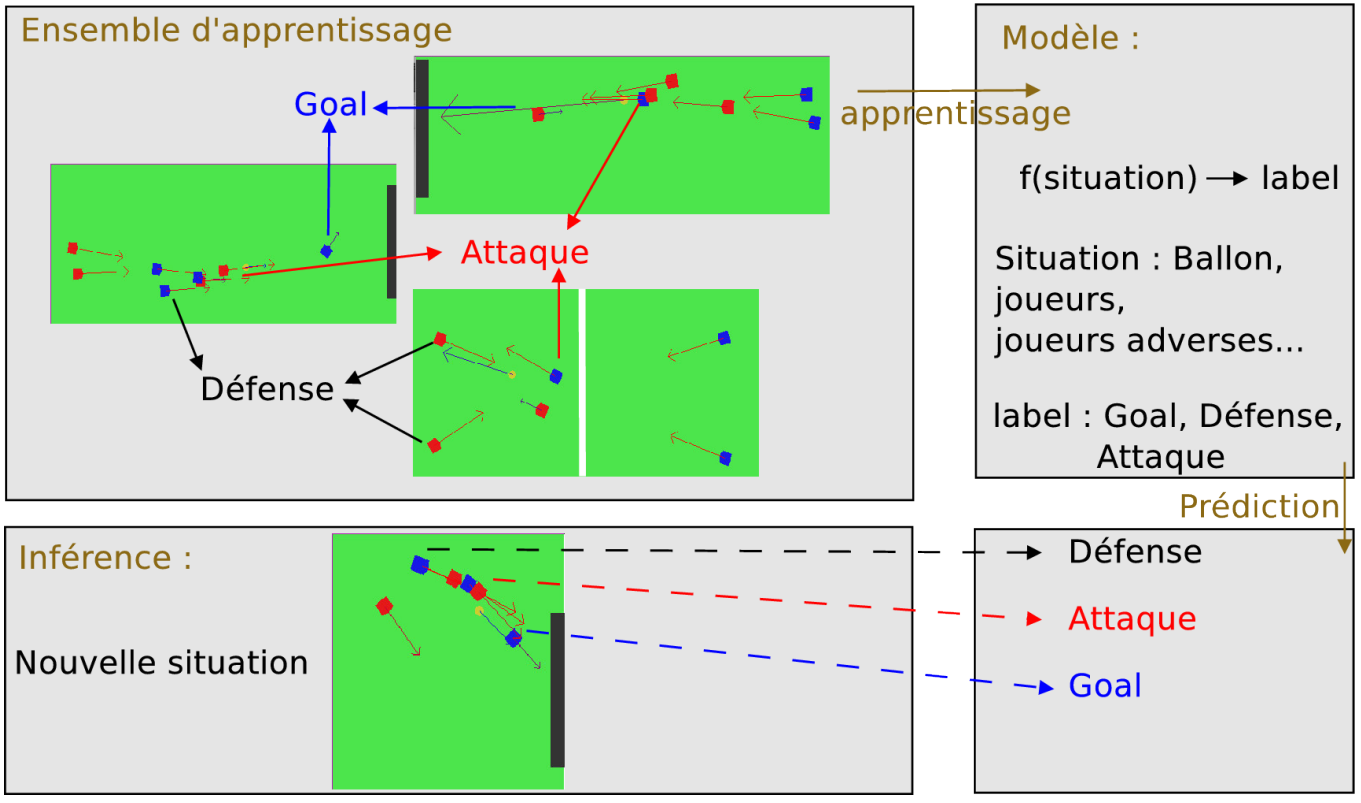


Principe

- Chaque nœud interne : un test sur une des dimensions de $\mathcal{X}$
- Chaque branche : un résultat du test
- Chaque feuille : un label de Y

$\Rightarrow$ classification en parcourant un chemin de la racine à une feuille.

Quels besoins ?



- Un ensemble d'apprentissage : des exemples de situations de jeu avec la bonne stratégie à appliquer
- Un espace de description : des attributs qui décrivent la situation de jeu

Ensemble d'apprentissage

Apprentissage par imitation

- Un prof $\rightarrow$ vous !
- des stratégies $\rightarrow$ les votres (simples ou complexes ?)
- des situations $\rightarrow$ jouer et choisisser les "bonnes" stratégies

Concrétement

- un nouvel interface disponible
- possibilité d'affecter une touche à un couple (stratégie,joueur)
- durant le jeu, à chaque fois qu'une nouvelle stratégie est choisie, enregistrement dans un fichier du couple `(state,stratégie)`

Description d'une situation

Génération d'attributs (*features*)

- Le choix des attributs est crucial pour un bon apprentissage
- Besoin de flexibilité : pouvoir en engendrer d'autres au fur et à mesure de l'avancement du projet

→ Ne pas reconstruire à chaque fois tous les exemples !

- Solution : une fonction de production d'attributs à partir d'un état
  `gen_feat(state,player,idteam)` $\rightarrow (x_1, x_2, \ldots, x_d)$

Sauver des objets dans un fichier : module `pickle`

- Module magique : permet de (presque) tout sauver (serialisation)
- Utilisation :

```
import pickle
#sauvegarde
with open("nom_fichier","wb") as f:
  pickle.dump(objet,f)
#chargement
with open("nom_fichier","rb") as :
  objet = pickle.load(f)
```

Exemple d'apprentissage d'un arbre

Module mathématique : numpy

```
import numpy as np
#creation de matrice 100x10
mat = np.zeros((100, 10))
#toutes les valeurs de la premiere colonne
mat[:,0]
#toutes les valeurs de la premiere ligne
mat[0,:]
```

Arbres de décision : module sklearn.tree

```
from sklearn.tree import DecisionTreeClassifier
#Creation de l'arbre
arbre = DecisionTreeClassifier()
#Apprentissage de l'arbre sur une matrice x (chaque ligne un exemple)
arbre.fit(x,y)
#prediction d'un exemple
arbre.predict(exemple)
#afficher un arbre
with open("tree.dot","w") as f:
     f = tree.export_graphviz(tr,out_file=f)
os.system("dot -Tpdf tree.dot -o tree.pdf")
```

Exemple d'utilisation

```
class ArbreStrategy(SoccerStrategy):
    #gen_feat  : fonction de generation de feature
    #a partir d'un state, player, teamid
    #tree : arbre de decision
    #dic_strat : dictionnaire de strategie
    #sur lequel l'arbre a ete appris
    def __init__(self, gen_feat,tree,dic_strat):
      self.name="Mon arbre"
      self.tree = tree
      self.gen_feat = gen_feat
      self.dic_strat=dic_strat
    def compute_strategy(self,idteam,idplayer,state):
      strat = self.tree.predict(self.gen_feat(idteam,idplayer,state))
      return dic_strat[strat].compute_strategy(idteam,idplayer,state)
```

Plan

Inspiration biologique

Le cerveau

- Robuste, tolérant aux fautes
- Flexible, sait s'adapter
- Gère les informations incomplètes
- Capable d'apprendre

Composé de neurones !

- 10^{11} neurones dans un cerveau humain
- 10^{4} connexions par neurones
- Potentiel d'action, neuro-transmetteurs, période réfractaire
- Signaux excitateurs / inhibiteurs

Problèmes

- Opacité des raisonnements
- Opacité des résultats

Inspiration biologique

Le cerveau

- Robuste, tolérant aux fautes
- Flexible, sait s'adapter
- Gère les informations incomplètes
- Capable d'apprendre

Composé de neurones !

- 10^{11} neurones dans un cerveau humain
- 10^{4} connexions par neurones
- Potentiel d'action, neuro-transmetteurs, période réfractaire
- Signaux excitateurs / inhibiteurs

Problèmes

- Opacité des raisonnements
- Opacité des résultats

Historique

Prémisses

- Mc Cullch et Pitts (1943) : 1er modèle de neurone formel. Base de l'IA
- Règle de Hebb (1949) : apprentissage par renforcement du couplage synaptique

Premières réalisations

- Adaline (Widrow-Hoff, 1960)
- Perceptron (Rosenblatt, 1958-1962)
- Analyse de Minsky et Papert (1969)

Développement

- Réseau bouclé (Hopfield 1982)
- Réseau multi-couches (1985)

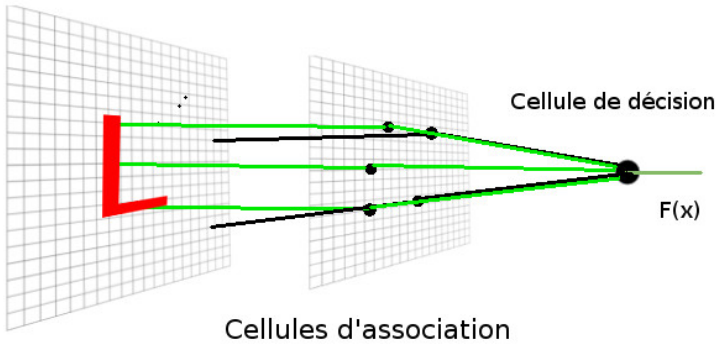
Deuxième renaissance

- Réseaux profonds (2000-)

Le perceptron de Rosenblatt (1960)

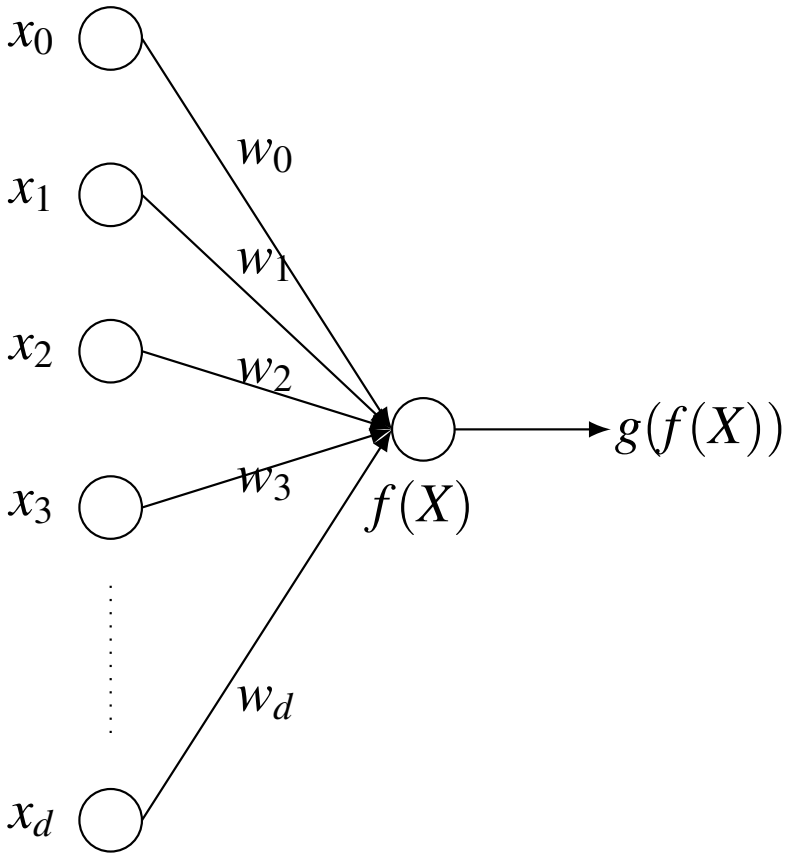
L'idée

- Reconaissance de forme (*pattern*) entre deux classes
- Inspirée cortex visuel



- Chaque cellule d'association produit une sortie $f_i(S)$ en fonction d'un stimulus
- La cellule de décision répond selon une fonction seuil $f_d(\sum w_i f_i(S_i))$

Formalisation



Le perceptron considère

- $f(\mathbf{x}) =< \mathbf{x}, \mathbf{w} >= \sum_{i=1}^{d} x_i w_i$
- Fonction de décision :
  $g(x) = sign(x)$

→ Sortie :
$g(f(\mathbf{x})) = sign(< \mathbf{x}, \mathbf{w} >)$

Considérations géométriques

Soit $y(x)$ la sortie attendue :

- Que représente w par rapport à la séparatrice ?
- Que représente $< w\mathbf{x} >$?
- Que représente $y(x) < w\mathbf{x} >$?
- A quoi correspond la règle de mise à jour :
  - Si $(y(x) < w.\mathbf{x} >) > 0$ ne rien faire
  - Si $(y(x) < w.\mathbf{x} >) < 0$ corriger $w = w + y(x)x$?

Algorithme de résolution

Algorithme du perceptron

- Initialiser au hasard w
- Tant qu'il n'y a pas convergence :
  - pour tous les exemples (x^i, y^i) :
    - si $(y^i < w.\mathbf{x}^i >) < 0$ alors $w = w + \epsilon y^i x^i$
- Décision : $f(x) = sign(< w\mathbf{x} >)$