Apprentissage à partir d'exemples

Plan

- Introduction à l'apprentissage
- Apprentissage inductif: Arbres de décision
- Apprentissage d'ensembles d'hypothèses
- Apprentissage à partir d'instances *: K*-voisins les plus proches
- Réseaux de Neurones

Apprentissage par observation

- Les perceptions ne devraient pas uniquement être utilisées pour choisir une action, mais aussi pour améliorer la capacité de l'agent à agir dans le futur.
- L'apprentissage est essentiel pour des environnements inconnus, où le concepteur ne peut pas tout savoir à l'avance.
- L'apprentissage modifie les mécanismes de décision de l'agent pour améliorer sa performance.

Apprentissage des composantes de l'agent

- Un agent peut apprendre à l'aide d'un retour approprié
  - Ex: conducteur de taxi.
    - À chaque fois que l'instructeur crie « Freine! », l'agent peut apprendre une règle condition-action.
    - En montrant à l'agent plusieurs images de caméra contenant des autobus, alors l'agent peut apprendre à reconnaître des autobus.
    - Il peut apprendre les effets de ses actions en les essayant. Ex: freiner sec sur une chaussée enneigée.
    - Si l'agent ne reçoit pas de pourboire pour un trajet où les passagers ont été brassés, alors l'agent pourrait apprendre une partie de sa fonction d'utilité globale.

Type de retour pour le critique

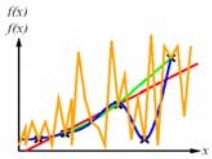
- Trois types d'apprentissage
  - Apprentissage supervisé
    - Dans ce cas, la base de données d'apprentissage est un ensemble de couples entrée-sortie (x_n, y_n) avec x_n 5 X et y_n 5 Y , que l'on considère être tirées selon une loi sur X x Y inconnue, par exemple x_n suit une loi uniforme et $y_n = f(x_n) + w_n$ où w_n est un bruit centré.
  - Apprentissage non supervisé
    - L'agent apprend à partir des relations entre les perceptions. Il apprend à prédire les perceptions à partir de celles du passé.
  - Apprentissage par renforcement
    - L'évaluation de l'action est faite par récompense ou punition (attachée à la tâche).

Apprentissage inductif

- Forme la plus simple d'apprentissage (supervisé) qui consiste à apprendre une fonction à partir d'exemples.
  - On veut apprendre une fonction *f*;
  - On a des exemples de la forme $(x, f(x))$;
  - On veut trouver une fonction h (hypothèse) qui se rapproche le plus possible de *f*.
- C'est un modèle très simplifié d'apprentissage
  - Ignore les connaissances a priori;
  - Suppose un environnement observable et déterministe;
  - Suppose que des exemples sont donnés.

Méthode d'apprentissage inductif

- h est cohérente si elle approxime f sur tous les exemples.
- Comment choisir alors parmi plusieurs hypothèses cohérentes?
  - Ex: Ajustement de courbes



• Laquelle choisir ?
• « Ockham's razor »: Choisir la plus simple parmi les hypothèses compatibles.
Si il n'y a pas d'hypothèse cohérente, il faut faire un compromis entre la cohérence et la simplicité.

Espace des hypothèses

- L'espace des hypothèses $\mathcal{H}$ est très important.
  - On doit choisir un espace d'hypothèses qui permet de représenter les données.
  - Par exemple, si on choisit des polynômes et que les données d'apprentissage proviennent d'une fonction sinusoïdale, on n'y arrivera pas.
  - Il ne faut pas prendre $\mathcal{H}$ trop grand, car l'apprentissage serait trop lent.

Espace des hypothèses (suite)

- Problème d'apprentissage est dit réalisable : Si l'espace des hypothèses $\mathcal{H}$ contient la vraie fonction.
- L'hypothèse la plus probable

$$h^* = \arg\max_{h \in \mathcal{H}} P(h|data)$$
$$h^* = \arg\max_{h \in \mathcal{H}} P(data|h)P(h)$$

- Dans ce cas, on peut favoriser via $P(h)$ les degré de polynôme 1, 2 ou autres.

Apprentissage d'arbres de décision

- Une des formes les plus simples d'apprentissage, mais tout de même une de celles qui connaissent le plus de succès.
- À partir d'exemples, le but est d'apprendre des structures d'arbres permettant de prendre des décisions.
- Chaque nœud représente un test à faire.
- Chaque branche représente une valeur possible résultant du test.
- Une feuille correspond à une prise de décision.

Exemple du restaurant

Faut-il ou pas attendre (*WillWait*) pour une table, dans un restaurant ?

Attributs

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range ($, $$, $$$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
10. *WaitEstimate*: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

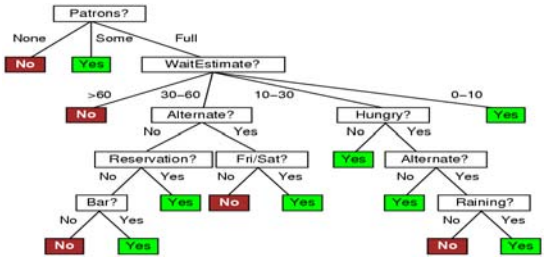
Exemple du restaurant

- Exemple: En arrivant au restaurant, est-ce que ça vaut la peine d'attendre ?

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Exemple du restaurant

- Exemple: En arrivant au restaurant, est-ce que ça vaut la peine d'attendre ?



Apprentissage d'arbres de décision

- Les arbres de décision sont des arbres qui permettent, la plupart du temps, de donner une réponse binaire du type oui ou non.
- Les instances sont représentés par des pairs (attribut, valeur). Les valeurs peuvent être discrètes ou continues.
- La fonction visée est généralement binaire, mais il est facilement possible d'étendre les arbres de décision pour qu'ils puissent avoir plusieurs valeurs en sorties.
- Ils pourraient donner en sortie des valeurs réelles, mais ce n'est pas beaucoup utilisé.

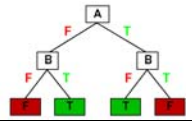
Apprentissage d'arbres de décision

- Les exemples d'entraînement peuvent contenir des erreurs.
  - Les erreurs dans les exemples d'entraînement sont souvent appelés des bruits.
  - Les arbres de décision sont robustes aux erreurs de classification et aux erreurs dans les valeurs des attributs.
  - Bien sûr, il ne faut pas qu'il y en ait trop.
- Il peut manquer des valeurs pour certains attributs dans les exemples d'entraînement.
- Les arbres de décision sont bons pour les problèmes de classification.

Expressivité

- Toutes les fonctions booléennes peuvent être représentées à l'aide d'un arbre de décision.
- On peut faire cela tout simplement en faisant correspondre chaque ligne de la table de vérité à un chemin dans l'arbre.
- Ceci va donner un arbre de grandeur exponentiel, mais on peut souvent faire beaucoup mieux.

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Expressivité

- Les arbres de décisions ne sont pas efficaces (grandeur exponentielle) pour représenter certaines fonctions, comme:
  - Fonction de parité: retourne 1 s'il y a un nombre pair de 1 dans les entrées.
  - Fonction de majorité: retourne 1 s'il y a plus de la moitié des entrées à 1.
- Combien d'arbres possibles pour n attributs booléens ?
  - Chaque réponse colonne peut être vu comme un nombre de 2^n
  - le nombre de table de vérité distinctes de 2^n rangées = 2^{2^n}

Avec 10 attributs booléens du restaurant, on a 2^{1024} soit environ 10^{308} *fonctions à choisir* !

Construire un arbre de décision

- L'apprentissage d'un arbre de décision est fait à partir d'exemples de valeurs d'attributs et de la valeur résultante du prédicat à apprendre.
- La valeur du prédicat à apprendre est appelée la classification de l'exemple (ex: Vrai / Faux).
- L'ensemble des exemples est appelé l'ensemble d'entraînement.

Construire un arbre de décision

- Le but est de trouver le plus petit arbre qui respecte l'ensemble d'entraînement.
- Il ne s'agit pas uniquement de mémoriser les observations: il faut trouver un arbre qui est capable d'extrapoler pour les exemples qu'il n'a pas déjà vus.
- L'arbre doit extraire des tendances ou des comportements à partir des exemples.

Algorithme

- Il construit les arbres de décision de haut en bas.
- Il place à la racine l'attribut le plus important, c'est-à-dire celui qui sépare le mieux les exemples positifs et négatifs.
- Par la suite, il y a un nouveau nœud pour chacune des valeurs possibles de cet attribut.
- Pour chacun de ces nœuds, on recommence le test avec le sous-ensemble des exemples d'entraînement qui ont été classés dans ce nœud.

Algorithme

function DECISION-TREE-LEARNING(*examples*, *attributes*, *default*) **returns** a decision tree
  inputs: *examples*, set of examples
    attributes, set of attributes
    default, default value for the goal predicate

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MAJORITY-VALUE(*examples*)
  else
    best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    tree ← a new decision tree with root test *best*
    m ← MAJORITY-VALUE($examples_i$)
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DECISION-TREE-LEARNING($examples_i$, *attributes* − *best*, *m*)
      add a branch to *tree* with label v_i and subtree *subtree*
    end
    return *tree*

Entropie pour le choix des attributs (Wiki)

- L'**entropie de Shannon**, due à Claude Shannon, est une fonction mathématique qui, intuitivement, correspond à la quantité d'information contenue ou délivrée par une source d'information.
- Cette source peut être un texte écrit dans une langue donnée, un signal électrique ou encore un fichier informatique quelconque (collection d'octets).

Entropie pour le choix des attributs (Wiki)

Pour une source (qui peut être une variable aléatoire discrète), X comportant n symboles, chacun ayant une probabilité P_i d'apparaître, l'**entropie** H de la source X est définie comme

$$H_b(X) = -\mathbb{E}[\log_b P(X = x_i)] = \sum_{i=1}^{n} P_i \log_b \left(\frac{1}{P_i}\right) = -\sum_{i=1}^{n} P_i \log_b P_i.$$

où $\mathbb{E}$ désigne l'espérance mathématique.

On utilise en général un logarithme à base 2 car l'entropie possède alors les unités de bits/symbole. Les symboles représentent les réalisations possibles de la variable aléatoire X.

$$H(X) = H_2(X) = -\sum_{i=1}^{n} P_i \log_2 P_i.$$

Choix de l'attribut

- On choisit l'attribut ayant le meilleur gain d'information:

$$Gain(S, A) \equiv Entropie(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropie(S_v)$$

$$Entropie(S) \equiv \sum_{i=1}^{c} (-p_i) log_2(p_i)$$

S : les exemples d'entraînement.
A : l'attribut à tester.
$V(A)$: les valeurs possibles de l'attribut A.
S_v : le sous-ensemble de S qui contient les exemples qui ont la valeur v pour l'attribut A.
c : le nombre de valeurs possibles pour la fonction visée.
p_i : la proportion des exemples dans S qui ont i comme valeur pour la fonction visée.

Exemple

- Est-ce une bonne journée pour jouer au tennis ?

Journée	Ciel	Température	Humidité	Vent	JouerTennis
J1	Ensoleillé	Chaude	Élevée	Faible	Non
J2	Ensoleillé	Chaude	Élevée	Fort	Non
J3	Nuageux	Chaude	Élevée	Faible	Oui
J4	Pluvieux	Tempérée	Élevée	Faible	Oui
J5	Pluvieux	Froide	Normal	Faible	Oui
J6	Pluvieux	Froide	Normal	Fort	Non
J7	Nuageux	Froide	Normal	Fort	Oui
J8	Ensoleillé	Tempérée	Élevée	Faible	Non
J9	Ensoleillé	Froide	Normal	Faible	Oui
J10	Pluvieux	Tempérée	Normal	Faible	Oui
J11	Ensoleillé	Tempérée	Normal	Fort	Oui
J12	Nuageux	Tempérée	Élevée	Fort	Oui
J13	Nuageux	Chaude	Normal	Faible	Oui
J14	Pluvieux	Tempérée	Élevée	Fort	Non

Exemple

- Premièrement, il faut choisir la racine de l'arbre.
- Pour cela, nous allons choisir l'attribut qui a le plus grand gain d'information.
  - Pour calculer le gain d'information, nous devons d'abord calculer l'entropie des exemples d'entraînement.
  - Il y a 9 exemples positifs et 5 exemples négatifs, donc nous obtenons une entropie de:

$$\begin{aligned} Entropie(S) &= \sum_{i=1}^{c}(-p_i)log_2(p_i) \\ &= (-9/14)\log_2(9/14) + (-5/14)\log_2(5/14) \\ &= 0.94 \end{aligned}$$

Exemple

- Maintenant, nous allons calculer le gain d'information pour le premier attribut, l'attribut *Ciel*.
  - Cet attribut a 3 valeurs possibles, donc les exemples d'entraînement seront regroupés en 3 sous-ensembles.
  - Nous commençons donc par calculer l'entropie des 3 sous-ensembles:

$$\begin{aligned} Entropie(S_{ensoleillé}) &= (-2/5)\log_2 2/5 + (-3/5)\log_2 3/5 = 0.971 \\ Entropie(S_{nuageux}) &= (-4/4)\log_2 4/4 + (-0/4)\log_2 0/4 = 0 \\ Entropie(S_{pluvieux}) &= (-3/5)\log_2 3/5 + (-2/5)\log_2 2/5 = 0.971 \end{aligned}$$

Exemple

- Le calcul du gain d'information pour l'attribut *Ciel* va donc donner:

$$\begin{aligned} Gain(S, Ciel) &= Entropie(S) - \sum_{v \in V(Ciel)} \frac{|S_v|}{|S|} Entropie(S_v) \\ &= 0.94 - ((5/14) \times Entropie(S_{ensoleillé}) + \\ &\quad (4/14) \times Entropie(S_{nuageux}) + \\ &\quad (5/14) \times Entropie(S_{pluvieux})) \\ &= 0.94 - ((5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971) \\ &= 0.94 - 0.694 \\ &= 0.246 \end{aligned}$$

Exemple

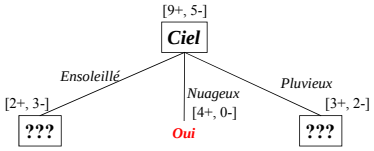
- On calcul le gain de la même manière pour les trois autres attributs:

$$\begin{aligned} Gain(S, Ciel) &= 0.246 \\ Gain(S, Humidité) &= 0.151 \\ Gain(S, Vent) &= 0.048 \\ Gain(S, Température) &= 0.029 \end{aligned}$$

- L'attribut qui a la plus grand gain d'information est l'attribut *Ciel*, donc se sera la racine de l'arbre de décision.

Exemple

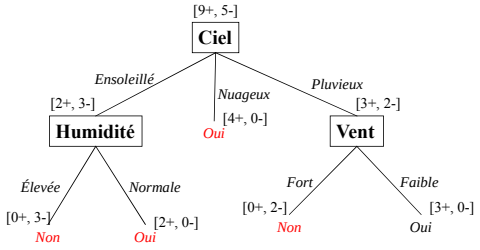
- En séparant les exemples selon les valeurs de l'attributs *Ciel*, on obtient l'arbre partiel:



- On peut voir que lorsque le ciel est nuageux, il reste uniquement des exemples positifs, donc ce nœud devient une feuille avec une valeur de *Oui* pour la fonction visée.
- Pour les deux autres nœuds, il y a encore des exemples positifs et négatifs, alors il faut recommencer le même calcul du gain d'information, mais avec les sous-ensembles restants.

Exemple

- En effectuant les calculs restants, on obtient:



Procédure générale d'apprentissage

1. Faire la collecte d'un grand ensemble d'exemples.
2. Diviser les exemples en deux ensembles: un d'entraînement et l'autre de test (2/3 et 1/3).
3. Utiliser l'ensemble d'entraînement comme exemples et générer l'hypothèse h.
4. Mesurer le pourcentage d'exemples de l'ensemble de test qui sont correctement identifiés par h.
5. Répéter les étapes 1 à 4 pour différentes tailles d'ensembles

Gérer les erreurs dans les exemples

- Si les exemples d'entraînement contiennent des erreurs, il se peut qu'on trouve un arbre plus grand, mais qui ne classifiera pas bien les instances autres que celles utilisées dans les exemples d'entraînement (surapprentissage).
- Pour vérifier notre arbre, on utilise un ensemble de validation (ou de test).
  - L'ensemble de validation est généralement le tiers des exemples disponibles pour l'apprentissage.

Élagage de l'arbre

- Cette méthode coupe des parties de l'arbre en choisissant un nœud et en enlevant tout son sous-arbre.
  - Ceci fait donc du nœud une feuille et on lui attribut la valeur de classification qui revient le plus souvent.
- Des nœuds sont enlevés seulement si l'arbre résultant n'est pas pire que l'arbre initial sur les exemples de validation.
- On continue tant que l'arbre résultant offre de meilleurs résultats sur les exemples de validation.
- Ceci a pour but de réduire l'arbre en enlevant des branches qui auraient été ajoutées par une erreur dans les exemples.

Exemple

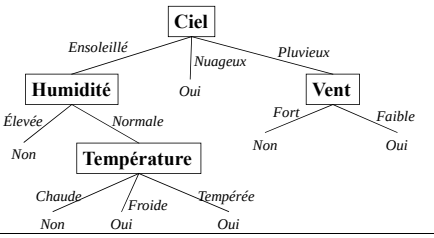
- Est-ce une bonne journée pour jouer au tennis ?

Journée	Ciel	Température	Humidité	Vent	JouerTennis
J1	Ensoleillé	Chaude	Élevée	Faible	Non
J2	Ensoleillé	Chaude	Élevée	Fort	Non
J3	Nuageux	Chaude	Élevée	Faible	Oui
J4	Pluvieux	Tempérée	Élevée	Faible	Oui
J5	Pluvieux	Froide	Normal	Faible	Oui
J6	Pluvieux	Froide	Normal	Fort	Non
J7	Nuageux	Froide	Normal	Fort	Oui
J8	Ensoleillé	Tempérée	Élevée	Faible	Non
J9	Ensoleillé	Froide	Normal	Faible	Oui
J10	Pluvieux	Tempérée	Normal	Faible	Oui
J11	Ensoleillé	Tempérée	Normal	Fort	Oui
J12	Nuageux	Tempérée	Élevée	Fort	Oui
J13	Nuageux	Chaude	Normal	Faible	Oui
J14	Pluvieux	Tempérée	Élevée	Fort	Non

Élagage de l'arbre

- Si on ajoute une erreur dans l'exemple précédent en modifiant le premier exemple, on obtient l'arbre suivant:

J1	Ensoleillé	Chaude	Élevée	Faible	Non
J1	Ensoleillé	Chaude	Normale	Faible	Non



Élagage de l'arbre

- L'élagage pour cet arbre consiste à enlever le nœud *Température* qui vient d'être ajouté avec l'erreur.
- Par la suite, on teste le nouvel arbre élagué sur les exemples de validation.
- Si la performance n'est pas diminuée, alors on conserve l'arbre élagué.

Valeurs d'attributs manquantes

- S'il y a des valeurs pour certains attributs qui ne sont pas disponibles, alors on peut :
  - Donner la valeur moyenne pour cet attribut.
    - On regarde les autres exemples et on calcule la moyenne des valeurs présentes.
    - On utilise cette moyenne pour estimer la valeur manquante dans le calcul du gain d'information.
  - Attribuer une probabilité pour la valeur manquante.
    - On regarde les autres exemples et on calcule la probabilité de chaque valeur possible pour l'attribut.
    - On utilise par la suite ces probabilités pour calculer le gain d'information.

Valeurs d'attributs manquantes

- Pour la première stratégie, les calculs ne changent pas. On ne fait qu'utiliser la valeur moyenne pour remplacer la valeur manquante.
- Pour la deuxième stratégie, les calculs sont modifiés pour utiliser les probabilités.
  - On commence par calculer la probabilité de chacune des valeurs possibles pour l'attribut manquant.
  - Par exemple, supposons qu'il nous manquerait une valeur pour l'attribut *Vent*.
    - La probabilité que le vent soit faible, selon nos exemples d'entraînement est de 8/14 * 100 = 57%.
    - La probabilité que le vent soit fort est de 6/14 * 100 = 43%.
  - Ceci nous donne des fractions d'exemples dans nos calculs.

Exemple

- Supposons que l'on a un exemple d'entraînement positif de plus, mais que la valeur pour l'attribut $Vent$ est absente.

$$
\begin{aligned}
Entropie(S) &= \sum_{i=1}^{c} -p_i log_2 p_i \\
&= -10/15 \log_2 10/15 + -5/15 \log_2 5/15 \\
&= 0.918 \\
Entropie(S_{fort}) &= (-4/6.43) \log_2 4/6.43 + (-3/6.43) \log_2 3/6.43 = 0.939 \\
Entropie(S_{faible}) &= (-7/8.57) \log_2 7/8.57 + (-2/8.57) \log_2 2/8.57 = 0.728 \\
Gain(S, Vent) &= Entropie(S) - \sum_{v \in V(Vent)} \frac{|S_v|}{|S|} Entropie(S_v) \\
&= 0.918 - ((6.43/15) \times Entropie(S_{fort}) + \\
&\quad (8.57/15) \times Entropie(S_{faible}) \\
&= 0.918 - ((6.43/15) \times 0.939 + (8.57/15) \times 0.728 \\
&= 0.918 - 0.818 \\
&= 0.1
\end{aligned}
$$

Attributs multivalués

- Il y a problème avec la fonction de gain d'information.
  - Lorsque les attributs ont beaucoup de valeurs possibles, comme par exemple un attribut date, leur gain est très élevé, car il classifie parfaitement les exemples.
  - Par contre, ils vont générer un arbre de décision d'une profondeur de 1 qui ne sera pas très bon pour les instances futures.
- Solution: on peut utiliser une fonction qui se nomme $GainRatio$ qui pénalise les attributs qui ont trop de valeurs possibles.

Attributs multivalués

- On utilise la fonction $GainRatio$ au lieu de $Gain$ lorsque l'on a des attributs multivalués.

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} log_2 \frac{|S_i|}{|S|}$$

S_1 à S_c sont les c ensembles résultant de la partition de S par les c valeurs de l'attribut A.

Attributs à valeurs continues

- On utilise un point de coupe pour obtenir une discrétisation des variables continues.
  - Ex: la variable $Température$ est continue et on a les 6 exemples suivants.

Température	40	48	60	72	80	90
JouerTennis	Non	Non	Oui	Oui	Oui	Non

  - On met les valeurs en ordre croissant et on regarde les endroits ou la fonction change de valeur.
  - À ces endroits, on choisit la médiane comme valeur de coupe.
  - On compare toutes les valeurs de coupe et on choisit celle qui apporte le plus grand gain d'information.

Exemple

Température	40	48	60	72	80	90
JouerTennis	Non	Non	Oui	Oui	Oui	Non

- Il y a deux valeurs de coupe:
  - (48 + 60)/2 = 54
  - (80 + 90)/2 = 85
- On calcul les gains d'information pour ces deux valeurs de coupes.

$$\begin{aligned} Entropie(S) &= \sum_{i=1}^{c} -p_i log_2 p_i \\ &= -3/6 \log_2 3/6 + -3/6 \log_2 3/6 \\ &= 1 \end{aligned}$$

Exemple

- Gain pour la coupe 54:

$$\begin{aligned} Gain(S, T_{54}) &= Entropie(S) - \sum_{v \epsilon V(T_{54})} \frac{|S_v|}{|S|} Entropie(S_v) \\ &= 1 - (((2/6) \times (-2/2 \log_2 2/2 + -0/2 \log_2 0/2)) + \\ &\quad ((4/6) \times (-3/4 \log_2 3/4 + -1/4 \log_2 1/4))) \\ &= 1 - (0 + 0.541) \\ &= 0.459 \end{aligned}$$

- Gain pour la coupe 85:

$$\begin{aligned} Gain(S, T_{85}) &= Entropie(S) - \sum_{v \epsilon V(T_{85})} \frac{|S_v|}{|S|} Entropie(S_v) \\ &= 1 - (((5/6) \times (-2/5 \log_2 2/5 + -3/5 \log_2 3/5)) + \\ &\quad ((1/6) \times (-1/1 \log_2 1/1 + -0/1 \log_2 0/1))) \\ &= 1 - (0.809 + 0) \\ &= 0.191 \end{aligned}$$

Exemple

- En comparant les gains d'informations, on se rend compte que 54 est la meilleure valeur de coupe.
  - Par conséquent, la température sera utilisée comme un attribut binaire qui ne pourra prendre que deux valeurs: plus petit que 54 ou plus grand ou égale à 54.

Exemples d'applications des arbres de décision

- Conception d'équipement de plate-forme de forage :
  - Un système expert appelé GASOIL (BP) pour la conception de système de séparation d'huile sur des plates-formes en mer.
  - La séparation dépend (entre autres) de : proportion relative de gaz, d'huile et d'eau, débit, pression, densité, viscosité, température, etc.
  - À ce moment, le plus gros système expert au monde: 2500 règles et un estimé de 10 personnes-années pour le construire.
  - Utilisation de l'apprentissage d'arbres de décision à partir d'une base de données d'exemples de conception existante: construit en 100 personnes-jours.
  - Plus performant que les humains et a sauvé des millions.

Exemples d'applications des arbres de décision

- Pilote d'avion: Une alternative aux méthodes formelles pour la construction d'un contrôleur pour des systèmes complexes est d'apprendre à poser les bonnes actions dans chaque situation.
- Pour des systèmes complexes, comme les avions, il peut être impossible de prendre l'approche formelle. Il reste l'autre approche…
  - 3 pilotes expérimentés avec plan de vol, répétés 30 fois: acquisition de données action-situation (90 000 exemples) avec 20 variables.
  - Apprentissage d'un l'arbre de décision.
  - Conversion en C et insertion dans la boucle de contrôle.
  - Résultat: Apprend à voler et mieux que les pilotes humains, car les erreurs humaines ont été éliminées par le mécanisme de généralisation.

Apprentissage d'ensembles d'hypothèses

- Au lieu d'apprendre une seule hypothèse, on en apprend plusieurs et on combine leur prédiction pour retourner la réponse.
  - Ex: si on a 5 arbres de décisions
    - Pour évaluer une instance, on exécute les 5 arbres
    - On retourne la réponse qui revient le plus souvent: Vote par majorité
    - Ex: (2 vrais, 5 faux), donc on retourne faux.

La méthode « boosting »

- Cette méthode est utilisée pour générer un ensemble d'hypothèses.
- Attribut un poids à tous les exemples.
- Plus un exemple a un grand poids, plus il sera considéré important par l'algorithme d'apprentissage.

La méthode « boosting »

- L'algorithme fonctionne de la manière suivante:

  Donner un poids de 1 à tous les exemples.
  Répéter M fois
  - Générer une hypothèse à partir des exemples
  - Augmenter le poids des exemples mal classifiés.
  - Diminuer le poids des exemples bien classifiés.

- L'hypothèse finale est la majorité pondérée des M hypothèses où le poids d'une hypothèse dépend de sa performance sur l'ensemble d'entraînement.

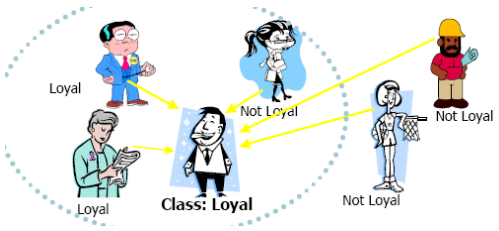
Apprentissage à base d'instances

Algorithme

- **Paramètre :** le nombre k de voisins
- **Donnée :** un échantillon de m exemples et leurs classes
  - La classe d'un exemple X est $c(X)$
- **Entrée :** un enregistrement Y
- 1. Déterminer les k plus proches exemples de Y en calculant les distances
- 2. Combiner les classes de ces k exemples en une classe c
- **Sortie :** la classe de Y est $c(Y) = c$

Exemple: Client loyal ou non

K = 3



Autres distances

Summation :

$$D(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Distance euclidienne ponderée :

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}$$

55

Exemple (1)

Customer	Age	Income	No. credit cards	Loyal
John	35	35K	3	No
Rachel	22	50K	2	Yes
Hannah	63	200K	1	No
Tom	59	170K	1	No
Nellie	25	40K	4	Yes
David	37	50K	2	?

56

Exemple (2)

K = 3

Customer	Age	Income	No. credit cards	Loyal	Distance from David
John	35	35K	3	No	sqrt [(35-37)²+(35-50)² +(3-2)²]=15.16
Rachel	22	50K	2	Yes	sqrt [(22-37)²+(50-50)² +(2-2)²]=15
Hannah	63	200K	1	No	sqrt [(63-37)²+(200-50)² +(1-2)²]=152.23
Tom	59	170K	1	No	sqrt [(59-37)²+(170-50)² +(1-2)²]=122
Nellie	25	40K	4	Yes	sqrt [(25-37)²+(40-50)² +(4-2)²]=15.74
David	37	50K	2	**Yes**	

57

Apprentissage à base d'instances

- Ne construit pas de fonctions à l'aide des exemples d'entraînement.
- Ne fait qu'emmagasiner les exemples.
- Le calcul n'est fait que lorsque le programme a une nouvelle instance à classer.
- Avantage: estimation locale pour chaque instance à classer.
- Désavantage: coût de classification élevé.

k-voisins les plus proches

- Les instances sont des points dans un espace à *n*-dimensions où n est le nombre d'attributs.
- On utilise la distance Euclédienne pour déterminer les voisins les plus proches.
- Une instance x est définie par son vecteur d'attributs:

$$< a_1(x), a_2(x), ..., a_n(x) >$$

- La distance entre deux instances x_i et x_j est :

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Fonction visée à valeurs discrètes

- L'ensemble V contient les valeurs possibles.
- L'estimation de la fonction f est tout simplement la valeur qui revient le plus souvent dans les *k* voisins.

$$\hat{f}(x_q) \leftarrow \max_{v \epsilon V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

où

$$\delta(a,b) = \begin{cases} 1 & \text{si } a = b \\ 0 & \text{sinon} \end{cases}$$

k-voisins les plus proches avec distances pondérées

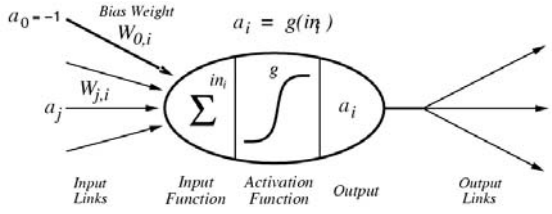
- Une amélioration à l'algorithme des k-voisins les plus proche est de pondérer la contribution de chacun des k voisins par leur distance par rapport à l'instance à classer.
- Les K-voisins sont robustes au bruit
- Pour les valeurs discrètes, on obtient:

$$\hat{f}(x_q) \leftarrow \max_{v \epsilon V} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \quad \text{où} \quad w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

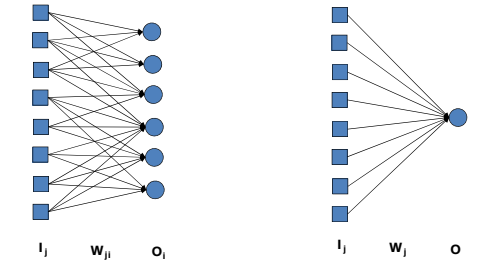
Réseaux de Neurones

Réseau de neurones

- Plusieurs neurones interconnectés ensemble
- Chaque neurone a la forme suivante:



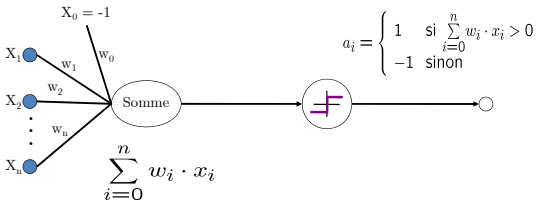
Perceptron



Réseau de perceptrons

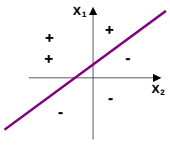
Un seul perceptron

Perceptron

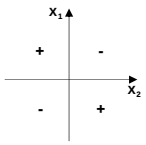


Perceptron

- Les perceptrons peuvent représenter toutes les fonctions linéairement séparables.



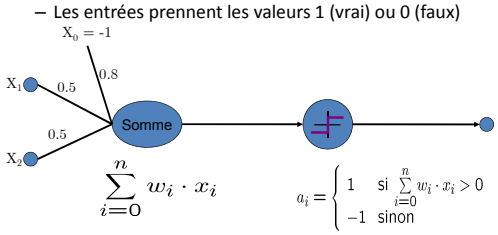
Fonction linéairement séparable



Fonction non linéairement séparable
(la fonction OU Exclusif)

Exemple de perceptron

- La fonction ET
  - Les entrées prennent les valeurs 1 (vrai) ou 0 (faux)



$$\sum_{i=0}^{n} w_i \cdot x_i \qquad a_i = \begin{cases} 1 & \text{si } \sum_{i=0}^{n} w_i \cdot x_i > 0 \\ -1 & \text{sinon} \end{cases}$$

Pour un OU, il suffirait de poser $w_0 = 0.3$

Règle d'apprentissage du perceptron

- Commencer avec des poids de valeurs aléatoires.
- Traiter tous les exemples d'entraînement jusqu'à ce que le perceptron les classifie tous correctement.
- À chaque exemple, les poids sont révisés à l'aide de la règle suivante:

$$w_i \leftarrow w_i + \Delta w_i$$
$$\Delta w_i = \eta(t - o)x_i$$

t : sortie désirée

o : sortie obtenue

η : **constante d'apprentissage, généralement petite (ex. 0.1)**

Règle d'apprentissage du perceptron

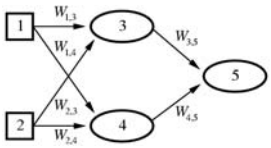
$$w_i \leftarrow w_i + \Delta w_i \quad \Delta w_i = \eta(t - o)x_i$$

- Pourquoi une telle règle converge-t-elle vers une bonne hypothèse ?

Sortie perceptron	$(t - o)$	Δw_i	w_i
Bonne valeur	0	0	Pareil
-1 au lieu de +1	+	+	Augmente
+1 au lieu de -1	-	-	Diminue

Ajustement des poids

- L'ajustement des poids dans les réseaux de neurones revient à modifier les paramètres de la fonction représentée par le réseau.
- Par exemple, la sortie du réseau a_5 dépend de la valeur des entrées a_1 et a_2. Les paramètres de la fonction sont les poids W.



$$a_5 = g(W_{3,5}a_3 + W_{4,5}a_4)$$
$$= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$$

Règle Delta

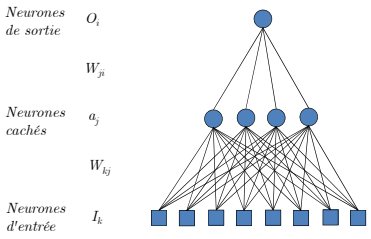
- Permet de gérer les exemples qui ne sont pas linéairement séparables
  - Converge vers l'estimation la plus juste possible
  - Utilise l'algorithme de la descente du gradient
  - La règle utilisée pour ajuster les poids est:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

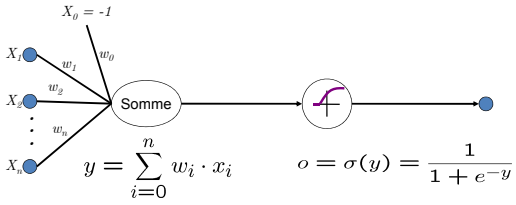
D : ensemble de tous les exemples d'entraînement

Réseau à plusieurs couches



Sigmoïd

- Fonction continue entre 0 et 1



$$y = \sum_{i=0}^{n} w_i \cdot x_i \qquad o = \sigma(y) = \frac{1}{1 + e^{-y}}$$

Algorithme de rétropropagation

- Utilise l'algorithme de descente du gradient.
- Tente de diminuer la différence au carré entre les sorties du réseau et les sorties désirées.
- L'erreur E du réseau est:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in sorties} (t_{kd} - o_{kd})^2$$

t_{kd} : **sortie désirée**
o_{kd} : **sortie obtenue**
D : **ensemble des exemples d'entraînement**
$sorties$: **ensemble des sorties du réseau**

Algorithme de rétropropagation
Texte du Pr. Marc Parizeau

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in sorties} (t_{kd} - o_{kd})^2$$

On exprime la variation de poids par
$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Texte du Prof Marc Parizeau

Algorithme de rétropropagation

- Initialiser les poids à de petites valeurs aléatoires (entre -0.05 et 0.05)
- Tant que la condition d'arrêt n'est pas atteinte
  - Pour chaque exemple d'entraînement
    - Calculer les sorties du réseau.
    - Pour toutes les unités de sortie k, calculer l'erreur:

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

    - Pour toutes les unités cachées h, calculer l'erreur:

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in sorties} w_{kh}\delta_k$$

    - Mettre à jour tous les poids w_{ji} :

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \eta \delta_j X_{ji}$$

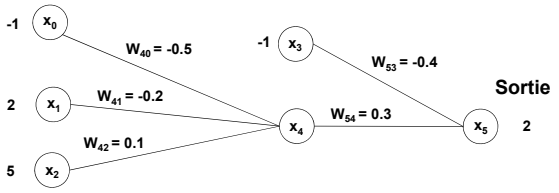
X_{ji} : l'entrée qui provient de l'unité i vers l'unité j

w_{ji} : le poids correspondant

Exemple

$\eta = 0.05$

Entrées



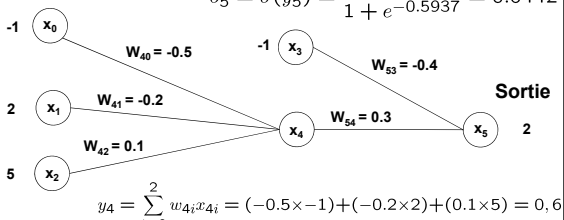
Exemple

$\eta = 0.05$

$$y_5 = \sum_{i=3}^{4} w_{5i}x_{5i} = (-0.4\times -1)+(0.3\times 0.6457) = 0,5937$$

Entrées

$$o_5 = \sigma(y_5) = \frac{1}{1+e^{-0.5937}} = 0.6442$$



$$y_4 = \sum_{i=0}^{2} w_{4i}x_{4i} = (-0.5\times -1)+(-0.2\times 2)+(0.1\times 5) = 0,6$$

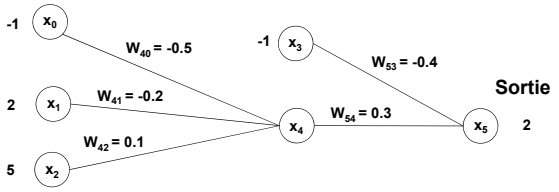
$$o_4 = \sigma(y_4) = \frac{1}{1+e^{-0.6}} = 0.6457$$

Exemple

$o_4 = 0.6457$
$o_5 = 0.6442$

$\eta = 0.05$

Entrées



$\delta_5 = o_5(1-o_5)(t_5-o_5) = 0.6442(1-0.6442)(2-0.6442) = 0.3108$

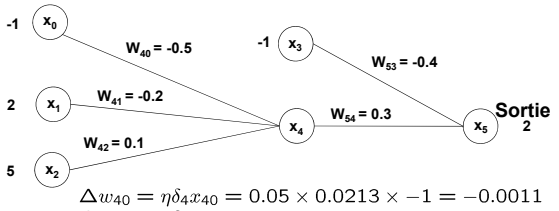
$\delta_4 = o_4(1-o_4) \sum_{k \in \{5\}} w_{k4}\delta_k = 0.6457(1-0.6457)(0.3 \times 0.3108) = 0.0213$

Exemple

$\eta = 0.05$

$\delta_4 = 0.0213 o_4 = 0.6457$
$\delta_5 = 0.3108 o_5 = 0.6442$

Entrées



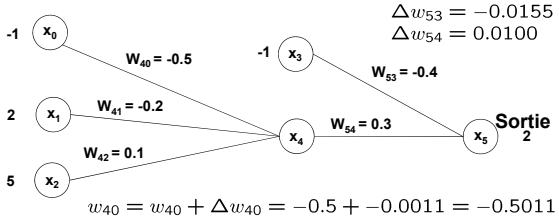
$\Delta w_{40} = \eta \delta_4 x_{40} = 0.05 \times 0.0213 \times -1 = -0.0011$
$\Delta w_{41} = \eta \delta_4 x_{41} = 0.05 \times 0.0213 \times 2 = 0.0021$
$\Delta w_{42} = \eta \delta_4 x_{42} = 0.05 \times 0.0213 \times 5 = 0.0053$
$\Delta w_{53} = \eta \delta_5 x_{53} = 0.05 \times 0.3108 \times -1 = -0.0155$
$\Delta w_{54} = \eta \delta_5 x_{54} = 0.05 \times 0.3108 \times 0.6457 = 0.0100$

Exemple

$\eta = 0.05$

$\Delta w_{40} = -0.0011$
$\Delta w_{41} = 0.0021$
$\Delta w_{42} = 0.0053$
$\Delta w_{53} = -0.0155$
$\Delta w_{54} = 0.0100$

Entrées



$w_{40} = w_{40} + \Delta w_{40} = -0.5 + -0.0011 = -0.5011$
$w_{41} = w_{41} + \Delta w_{41} = -0.2 + 0.0021 = -0.1979$
$w_{42} = w_{42} + \Delta w_{42} = 0.1 + 0.0053 = 0.1053$
$w_{53} = w_{53} + \Delta w_{53} = -0.4 + -0.0155 = -0.4155$
$w_{54} = w_{54} + \Delta w_{54} = 0.3 + 0.0100 = 0.3100$

Condition d'arrêt

- Le nombre d'itérations est important car:
  - Si trop faible, l'erreur n'est pas suffisamment réduite.
  - Si trop grand, le réseau devient trop spécifique aux données d'entraînement.
- Il y a plusieurs conditions d'arrêt possible:
  - Après un certain nombre fixe d'itérations.
  - Lorsque l'erreur dans les sorties des exemples d'entraînement descend en dessous d'une certaine borne.
  - Lorsque l'erreur avec les exemples de validation descend en dessous d'une certaine borne.

Convergence

- La surface qui représente l'erreur contient plusieurs minimums locaux.
- L'algorithme ne garantit pas l'atteinte du minimum global.
- Mais, tout de même **bon en pratique**
  - Poids initialiser à de petites valeurs.
  - La sigmoïd est presque linéaire pour de petites valeurs, donc pas de minimums locaux.
  - Les poids grandissent à une vitesse contrôlée par le coefficient d'apprentissage.
  - Donc, les poids ont le temps de se rapprocher du minimum global avant que la fonction devienne plus complexe avec plusieurs minimums locaux.

Caractéristiques

- Instances représentées par des pairs attribut-valeur. Les attributs peuvent être indépendants ou non. Les valeurs peuvent être réelles.
- Les sorties peuvent être des valeurs discrètes, continues ou vectorielles.
- Les exemples peuvent contenir des erreurs.
- Le temps d'apprentissage peut être long.
- L'évaluation de la fonction apprise est très rapide.
- La fonction visée est souvent très peu compréhensible par un humain.

Construction du réseau

- Il existe aucune techniques précise
  - La construction se fait souvent par essaies et erreurs
- Il y a quelques algorithmes:
  - Algorithmes génétiques
    - Demande beaucoup de temps de calcul
  - « Optimal brain damage »
    - Commence avec un réseau complètement connecté et on enlève les poids qui sont proches de zéro. On peut aussi enlever les neurones qui ne contribuent pas au résultat.
  - « Tiling algorithm »
    - Commence avec un neurone et ajoute des neurones jusqu'à ce qu'ils couvrent tous les exemples d'entraînement.

Applications

- L'interprétation d'images.
- La reconnaissance vocale.
- La reconnaissance de mots écrits à la main.

  http://members.aol.com/Trane64/java/JRec.html
- L'apprentissage de stratégies de contrôle pour les robots.
- Une des meilleurs méthodes connues pour l'interprétation de données provenant de capteurs dans le monde réel.

Applications

- Le système ALVINN a conduit une voiture à 55 milles/h pendant 90 miles sur une autoroute.

Applications

- L'architecture d'ALVINN est un réseau de neurones à rétropropagation avec une couche cachée.
- La sortie est une distribution de Gauss centrée autour de la direction appropriée.