Apprentissage automatique

Damien Nouvel

Plan

Généralités sur l'apprentissage

- Qu'est-ce que l'**apprentissage**
  - Pour l'**humain**, acquisition
    - De **connaissances** (cognition, par ex. une langue)
    - D'un **savoir-faire** (tâche, par ex. marcher)
  - Pour l'**intelligence artificielle**
    - Prendre une décision *intellligente*
    - Construire un modèle **automatiquement**

  $\Rightarrow$ Difficile à déterminer ex-nihilo

  $\Rightarrow$ Importance de définir **les données** et l'**objectif**
- Notion d'**erreur**
  - Permet de **guider** l'apprentissage
  - **Objectif** : avoir le moins d'erreurs possible
  - Lien avec la fonction d'**évaluation**

Généralisation vs spécialisation

- Tendances générales
  - Généralisation
    - Prise en compte de données plus diverses
    - Adaptation aux nouvelles données
    - ⇒ Perte en précision
    - ⇒ Gain en rappel
    - ⇒ **Sous-apprentissage**
  - Spécialisation
    - Focalisation sur des données spécifiques
    - Difficulté face aux nouvelles données
    - ⇒ Gain en précision
    - ⇒ Perte en rappel
    - ⇒ **Sur-apprentissage**

⇒ Plus de données d'entraînement peuvent généraliser ou spécialiser

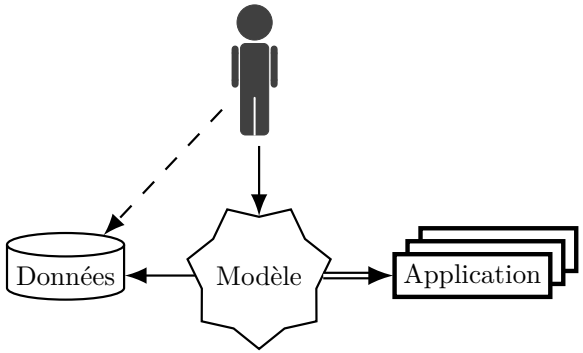
⇒ Attention aux effets de bord (données éparses)

Degré de supervision

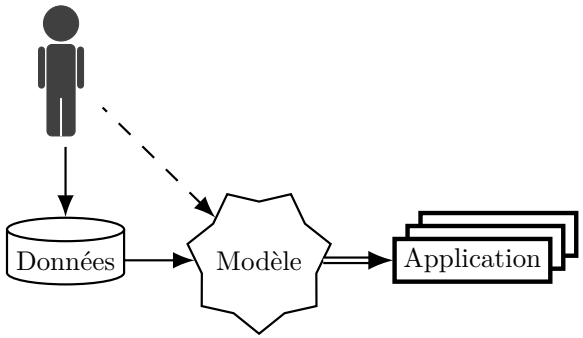
- Effort sur le coût de **conception** (ingénierie) des logiciels

⇒ Degré d'intervention de l'humain dans la conception

- Schématiquement
  - Conception manuelle (symbolique, à base de règles) :



  - Conception automatique (statistique, guidé par les données) :



Méthodologie supervisée

- Pour un jeu de données
  - **Entraîner**, **développer**, **évaluer** le(s) modèle(s)
  - Attention au **sur-apprentissage** !

  ⇒ La connaissance du jeu de données **biaise** la modélisation

  ⇒ Par exemple, apprentissage **par cœur**
- Jeux de données pour **apprentissage supervisé**
  - **Entraînement** (train) : stat. pour les modèles (probabilités)
  - **Développement** (dev) : amélioration itérative des modèles
  - **Evaluation** (test) : test final (idéalement unique)

⇒ Utilisation de **validation croisée** (cross-validation)

- Diviser le jeu de données D en n sous-ensembles (souvent 10)
- Pour chaque sous-ensemble i :
  - Paramétrer le modèle sur $D - d_i$
  - Evaluer le modèle sur d_i

- Méthodes non-supervisées : pas de données d'entraînement !

Exploitation des données

- Formalisation des données
  - Ensemble de données en **entrée** $X = \{x_1, x_2, x_3 \dots x_n\}$
    - Généralement non-ordonné
    - Chaque exemple est une structure (vecteur)
    - **Caractéristiques** / **features** $x_{i1} \dots x_{in}$

    $\Rightarrow$ Forme **matricielle** : matrice (lignes x_{i*} / colonnes x_{*j})
  - Ensemble de données en **sortie** $Y = \{y_1, y_2, y_3 \dots y_n\}$
    - Résultat attendu associé à chaque x_i correspond un y_i
    - Généralement une seule valeur (booléenne, entière ou réelle)

    $\Rightarrow$ Prédiction : **régression** (continue) / **classification** (classes)

    $\Rightarrow$ Forme **matricielle** : vecteur (une colonne)
- Tenir compte des données en entrée X
  - Telles quelles : apprentissage par cœur, KNN
  - Extraction de statistiques : TF.IDF, Bayes
  - Itération sur les données : neurones, logit, SVM, CRF
- Tenir compte des données en sortie Y : degré de **supervision**

Outils pour l'apprentissage automatique

- Quelques librairies disponibles
  - Weka : `https://www.cs.waikato.ac.nz/~ml/weka`
  - scikit-learn : `http://scikit-learn.org`
  - TensorFlow (Google) : `https://www.tensorflow.org`
  - OpenAI : `https://openai.com`
  - Keras : `https://keras.io`
  - Caffe : `http://caffe.berkeleyvision.org`
  - Torch : `http://torch.ch`
  - Azure (MS) : `https://azure.microsoft.com`

Plan

K plus proches voisins

- Les données X sont utilisées sans transformations
- Pour un **nouvel exemple** z à classifier
  - Calcul de la **distance** entre z et chaque x_i
    - Distance de Manhattan : $D(x_i, z) = \sum_j |x_{ij} - z_j|$
    - Distance euclidienne : $D(x_i, z) = \sqrt{\sum_j (x_{ij} - z_j)^2}$
    - Similarité cosinus : $D(x_i, z) = \frac{x_i \cdot z}{\|x_i\| * \|z\|} = \frac{\sum_j x_{ij} * z_j}{\sqrt{\sum_j x_{ij}^2} * \sqrt{\sum_j z_j^2}}$
    - Distance de Jaccard : $D(x_i, z) = 1 - \frac{|x_i \cap z|}{|x_i \cup z|}$
  - Sélection des K exemples x_i les plus proches (distance)
  - Choix de la classe la plus fréquente parmi les K

$\Rightarrow$ **Avantages :** résultats honorables pour peu de calculs

$\Rightarrow$ **Inconvénients :** il faut fixer K

- En anglais : K Nearest Neighbours (KNN)

TF.IDF

- Algorithme tourné vers la **recherche d'informations**
  - Caractéristiques x_{ij} : **termes** dans les **documents**
  - Matrice termes / documents (*vector space model*)
- Apprentissage du modèle
  - Poids des termes dans la collection d'exemples X :

$$idf_j = \frac{|X|}{|\{x_i | x_{ij} > 0\}|}$$

  - Vecteur représentant chaque exemple x_i^t : $x_{ij}^t = x_{ij} * idf_j$
- Pour un **nouvel exemple** z à classifier
  - Vecteur z^t représentant le nouvel exemple : $z_j^t = z_j * idf_j$
  - Calcul des distances (similarité cosinus) entre z^t et chaque x_i^t

⇒ **Avantages :** bonne prise en compte des termes spécifiques

⇒ **Inconvénients :** fortement dépendant des termes présents

⇒ **LSA** (sémantique latente) : termes → concepts → documents

Classifieur bayésien naïf : principe

- Discret : probabilité de $P(y_i)$ dépend des **facteurs** x_{ij}
  - Formulation mathématique :

$$P(y_i|x_{i1} \dots x_{in}) * P(x_{i1} \dots x_{in}) = P(x_{i1} \dots x_{in}|y_i) * P(y_i)$$

$$P(y_i|x_{i1} \dots x_{in}) = P(y_i) * \frac{P(x_{i1} \dots x_{in}|y_i)}{P(x_{i1} \dots x_{in})}$$

    - $P(y_i|x_{i1} \dots x_{in})$: probabilité **a posteriori** (postérieure)
    - $P(y_i)$: probabilité **a priori** (antérieure)
    - $P(x_{i1} \dots x_{in}|y_i)$: **vraisemblance**
    - $P(x_{i1} \dots x_{in})$: **évidence** (proba jointe)

  $\Rightarrow$ Trouver le y_i qui maximise ce calcul
    - La quantité $P(x_{i1} \dots x_{in})$ est **constante**
    - Calcul de $P(y_i)$ sans difficultés
    - Comment calculer $P(x_{i1} \dots x_{in}|y_i)$?

Classifieur bayésien naïf : hypothèse naïve

⇒ Données trop **éparses** pour l'évènement joint $x_{i1} \dots x_{in}$

- Remarque : si les x_{ij} étaient indépendants deux à deux :

$$P(y_i|x_{i1} \dots x_{in}) = P(y_i) * \prod_j \frac{P(x_{ij}|y_i)}{P(x_{ij})}$$

- Probabilités conditionnelles : $P(A, B|C) = P(A|C) * P(B|C, A)$

$$\begin{aligned}
&P(x_{i1} \dots x_{in}|y_i) \\
&= P(x_{i1}|y_i) * P(x_{i2} \dots x_{in}|y_i, x_{i1}) \\
&= P(x_{i1}|y_i) * P(x_{i2}|y_i, x_{i1}) * P(x_{i3} \dots x_{in}|y_i, x_{i1}, x_{i2}) \\
&= P(x_{i1}|y_i) * P(x_{i2}|y_i, x_{i1}) \cdots * P(x_{in}|y_i, x_{i1} \dots x_{in-1})
\end{aligned}$$

⇒ Hypothèse naïve : $P(x_{ij}|y_i, x_{i1} \dots x_{ij-1}) \approx P(x_{ij}|y_i)$

$$\begin{aligned}
&= P(x_{i1}|y_i) * P(x_{i2}|y_i) * P(x_{i3}|y_i) \dots P(x_{in}|y_i) \\
&= \prod_j P(x_{ij}|y_i)
\end{aligned}$$

Classifieur bayésien naïf : facteurs

- Calcul de la probabilité des classes selon
  - La probabilité **antérieure** de la classe $P(y_i)$
  - La **vraisemblance** des features sachant la classe $P(x_{ij}|y_i)$
- Pour un **nouvel exemple** z à classifier, pour chaque classe y
  - Calcul de $P(y_i) * \prod_j P(z_j|y_i)$
  - Sélection de la classe y_i qui maximise ce calcul

$\Rightarrow$ **Avantages :** tient compte de nombreuses caractéristiques

$\Rightarrow$ **Inconvénients :** ne calcule pas des probabilités

$$\Rightarrow P(y_i|x_{i1} \dots x_{in}) = \frac{P(y_i) * \prod_j P(z_j|y_i)}{P(y_i) * \prod_j P(z_j|y_i) + P(\overline{y_i}) * \prod_j P(z_j|\overline{y_i})}$$

Moindres carrés : formulation

- Fonction **linéaire** de prédiction
  - calcul sur les X pour obtenir les Y
  - Introduction de **poids** w

    $f(x_i, w) = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} \cdots + b$

    $= \sum_j w_j x_{ij} + b$

  ⇒ Ajuster w de manière à ce que $\forall i : f(x_i, w) \approx y_i$
- Fonction d'**erreur**
  - **Distance** entre $f(x_i, w)$ et y_i
  - Erreur quadratique **moindres carrés** : $E = \sum_i (y_i - f(x_i, w))^2$

  ⇒ À minimiser : $w^*, b^* = \text{argmin}_{(w,b)} E$
- Pour une seule composante
  - $E = \sum_i (y_i^2 + w_1^2 x_{i1}^2 + b^2 - 2 y_i w_1 x_{i1} - 2 y_i b + 2 w_1 x_{i1} b)$
  - Dérivées
    - $\nabla_{w_1} E = \sum_i (2 w_1 x_{i1}^2 - 2 y_i x_{i1} + 2 x_{i1} b)$
    - $\nabla_b E = \sum_i (2b - 2 y_i + 2 w_1 x_{i1})$

Moindres carrés : résolution

- Forme **matricielle** du problème
  - X est la matrice des x_i

  $\Rightarrow$ On ajoute à X une colonne telle que $x_{*0} = 1$
  - W est la matrice colonne des poids

  $\Rightarrow$ On ajoute à W une colonne telle que $w_0 = b$
  - Y est la matrice colonne des y_i

$\Rightarrow$ Il faut trouver W^* qui minimise $\| Y - X \cdot W \|^2$ (convexe)

- **Solution**
  - Optimum global : $\nabla \| Y - X \cdot W^* \|^2 = 0$
  - Équations normales : $X^t X W^* = X^t Y$

  $\Rightarrow$ $W^* = (X^t X)^{-1} X^t Y$

$\Rightarrow$ **Avantages :** calcul non itératif

$\Rightarrow$ **Inconvénients :** calcul coûteux

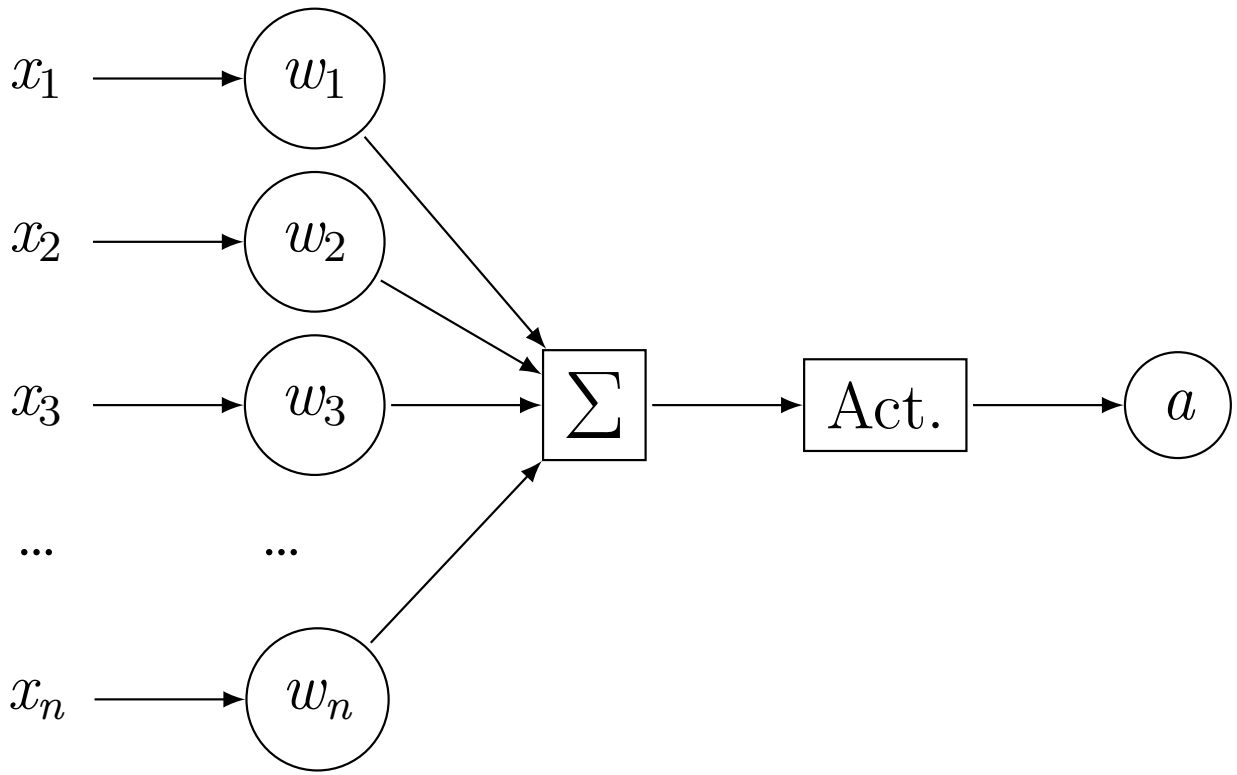
Plan

Réseaux de neurones

- Inspirés des **neurones biologiques**
  - **Dendrites** : entrées, reçoivent de l'information
  - **Noyau** : activé à partir d'un certain potentiel d'activité
  - **Axone** : sortie, conduit les informations
  - Chaque neurone fait le lien entre entrée(s) et sortie(s)
- Généralités sur les réseaux de neurones
  - Neurone **artificiel** / formel (W. McCulloch, W. Pitts, 1960)
  - Couche de neurones entre les X et les Y
  - Combinaison des X par un vecteur de poids w
  - Apprentissage **itératif** sur les données
  - Notion d'**activation**

Perceptron : principe

- Réseau de neurones simple (F. Rosenblatt, 1957)
  - Une seule couche
  - Linéaire



Perceptron : activation

- Combinaison des entrées comme **fonction**
  - **Pondération** des variables en entrée

  $\Rightarrow$ $z_i = \sum_j x_{ij} w_j + b$

  $\Rightarrow$ Valeur réelle : $-\infty < z_i < +\infty$
- Règle d'**activation** de Heaviside
  - Sortie attendue du neurone : 1 (activé) 0 (non-activé)
  - $a_i = \begin{cases} 1 & \text{si } z_i \geqslant 0 \\ 0 & \text{si } z_i < 0 \end{cases}$

$\Rightarrow$ Selon les **entrées** et les **poids**, le neurone est activé ou non

Perceptron : règle d'apprentissage

- Apprentissage **itératif**
  - Prise en compte des exemples **un à un**
  - Calcul de l'erreur $\delta_i = y_i - a_i$
    - 0 si $y_i - a_i$ (tous deux à 0 ou tous deux à 1)

    $\Rightarrow$ Pas d'erreur, on ne touche à rien
    - $+1$ si $y_i = 1$ et $a_i = 0$

    $\Rightarrow$ Erreur, pas activé : il faut augmenter w
    - -1 si $y_i = 0$ et $a_i = 1$

    $\Rightarrow$ Erreur, activé : il faut diminuer w
  - Modification de chaque poids : $w'_j = w_j + \alpha \delta_i x_{ij}$
    - α : **pas d'apprentissage** (entre 0 et 1, généralement 0.1)
    - δ_i : l'erreur guide la direction de la modification
    - x_{ij} : présence de la composante j pour chaque exemple

$\Rightarrow$ Jusqu'à convergence (ou nombre max. d'itérations)

Régression logistique : modélisation

- Modèle **exponentiel** : **maxent**, **logit**, **softmax**
- Exemple (Berger, 1996) : quelle traduction pour le mot *in* ?
  - Liste de traductions :
    dans, *en*, *à*, *au cours de*, *pendant*
  - Il faut choisir un élément dans la liste :
    $P(dans) + P(en) + P(a) + P(aucoursde) + P(pendant) = 1$

  $\Rightarrow$ Une infinité de solutions possibles ...
  - Choix arbitraire : $P(dans) = 1$
  - Modélisation équiprobable (loi uniforme) : $P(X) = 1/5$

Régression logistique : statistiques

- Introduction de données statistiques
  - Expert : *dans* ou *en* 30% du temps :
    $P(dans) + P(en) = 3/10$
    $P(dans) + P(en) + P(a) + P(aucoursde) + P(pendant) = 1$
  - Modélisation équiprobable (loi uniforme) :
    $P(dans) = P(en) = 3/20$
    $P(a) = P(aucoursde) = P(pendant) = 7/30$
  - Expert : *dans* ou *à* 50% du temps :
    $P(dans) + P(en) = 3/10$
    $P(dans) + P(en) + P(a) + P(aucoursde) + P(pendant) = 1$
    $P(dans) + P(a) = 1/2$

  $\Rightarrow$ Quelles probabilités uniformes ?!

Régression logistique : distribution jointe

- Selon le jeu de données $\tilde{P}(x, y)$
  - x : phrases à traduire contenant *in*
  - y : traductions possibles de *in*

  $\Rightarrow$ $\tilde{P}(x, y) = freq(x, y)/N$ (avec N le nombre d'observations)
- Fonction d'indicateur (caractéristique, feature)
  - $f(x, y) =$
    - 1 si $y =$ *en* et *April* suit *in* dans x
    - 0 sinon

  $\Rightarrow$ Indique les situation d'un contexte et d'une traduction
  - Probabilité empirique : $\tilde{P}(f) = \sum_{x,y} \tilde{P}(x, y) f(x, y)$

  $\Rightarrow$ Peut être modélisée pour n'importe quelle feature
  - Estimation : $P(f) = \sum_{x,y} \tilde{P}(x) P(y|x) f(x, y)$

  $\Rightarrow$ Modélisation de $P(f)$ selon le paramètre $P(y|x)$

  $\Rightarrow$ On souhaite faire en sorte que $\tilde{P}(f) = P(f)$

Régression logistique : entropie conditionnelle

- Chaque feature f_i donne une contrainte à satisfaire
- Distribution uniforme : entropie conditionnelle sur $P(y|x)$
  - $H(P) = -\sum_{x,y} \tilde{P}(x) P(y|x) log(P(y|x))$

  $\Rightarrow$ Déterminer $P(y|x)$ qui maximise cette quantité

  $\Rightarrow$ Modèle le moins arbitraire
- Problème contraint
  - $P(y|x) > 0$
  - $\sum_y P(y|x) = 1$ quelque soit x
  - $\sum_{x,y} \tilde{P}(x) P(y|x) f_i(x,y) = \sum_{x,y} \tilde{P}(x,y) f_i(x,y)$

$\Rightarrow$ Pas de calcul direct / solution exacte

Régression logistique : poids du modèle

- Lien entre maximum d'entropie et maximum de vraisemblance
- Utilisation du lagrangien pour trouver la dérivée ...
  - Introduction de paramètres (poids) w_i

$$P(y|x) = \frac{exp(\sum_i w_i f_i(x,y))}{\sum_y exp(\sum_i w_i f_i(x,y))}$$

  - Itérations (gradient) pour optimiser les poids

$$w^* = argmax_w \sum_{x,y} \tilde{P}(x,y) log(P(y|x))$$

$\Rightarrow$ Algorithmes : GIS, IIS, L-BFGS ...

Plongements de mots

- « You shall know a word by the company it keeps » (Firth, 57)
- Analyse « distributionnelle » des mots en contexte (Harris, 60)

⇒ Des mots **similaires** partagent les même **contextes**

- Plongements (*embeddings*) (Collobert, 2011)
- Word2Vec **non-supervisé** (Mikolov et. al., 2012)
  - Prédire un mot selon son contexte : **CBOW** (vs SkipGram)
  - Modèle neuronal (poids) avec softmax (et *negative sampling*)

  ⇒ Poids : **dimensions** ($\approx$ 100-500)

  ⇒ Similarités : $bicyclette \approx velo$

  ⇒ Analogies : $roi - homme + femme \approx reine$
- Limites de l'algorithme initial
  - Niveau token : pas d'expressions polylexicales
  - Pas de morphologie : déclinaisons / conjugaisons ?

  ⇒ FastText (Bojanowski, 2016)
  - Similarités **syntaxique et sémantique** ?