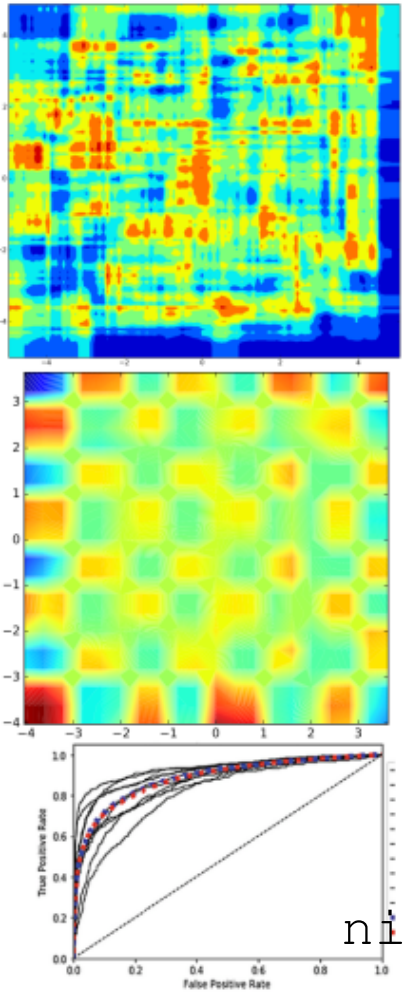


Théorie de l'apprentissage
Ensemble learning
Mesures d'évaluation
Multi-classe

Cours 7
Machine Learning
Master DAC

Nicolas Baskiotis
nicolas.baskiotis@sorbonne-universite.fr

équipe MLIA, Institut des Systèmes Intelligents et de Robotique (ISIR)
Sorbonne Université

S2 (2022-2023)

Plan

Apprentissage supervisé et risque

Problèmatique de l'apprentissage supervisé

- un ensemble d'apprentissage $E = \{(\mathbf{x}^i, y^i)\} \in \mathcal{X} \times \mathcal{Y}$
- un ensemble de fonctions $\mathcal{F}$
- un coût $\ell(\hat{y}, y) : Y \times Y \to \mathbb{R}^+$
- trouver $f = argmin_{f \in \mathcal{F}} \sum_i \ell(f(\mathbf{x}^i), y^i)$

Minimisation du risque et risque bayésien

- Risque : $R_{\ell,p}(f) = \mathbb{E}_{\mathbf{x},y}[\ell(y, f(\mathbf{x})] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(\mathbf{x}), y) p(\mathbf{x}, y) d\mathbf{x} dy$
- Risque bayésien : $R^\star_{\ell,p} = min_{f:\mathcal{X}\to\mathcal{Y}} \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(\mathbf{x}), y) p(\mathbf{x}, y) d\mathbf{x} dy$
- Risque empirique sur n échantillons : $\hat{R}_{n,\ell} = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}^i), y^i)$
- Objectif : trouver une fonction dont le risque est proche de l'optimal.

Consistance d'un algorithme

Définition

- Un algorithme produit une fonction f en fonction d'un jeu de données E de taille n.
- Un algorithme est **universellement consistant** si pour toute distribution p des données le risque de la fonction apprise converge vers le risque bayésien quand $n \to \infty$: $\quad lim_{n\to\infty} R_{n,\ell,p} \to R^*_{\ell,P}$

Théorème de Stone, 1977

- Sous certaines conditions, pour certaines fonctions de coût, les algorithmes vu dernièrement sont universellement consistants.
- Mais on dispose rarement d'une infinité de données . . .
- Et à quel rythme on converge ?

No free lunch

Théorème Devroy, 1982

Pour tout algorithme universellement consistant et pour tout taux de convergence a_n, il existe une distribution p telle que le taux de convergence de l'algorithme soit plus lent que a_n.

Autrement dit

- Pour deux algorithmes d'apprentissage 1 et 2, sans a priori sur le problème, à n fixé :
  - si toutes les fonctions cibles sont équiprobable, en espérance de l'erreur sur tous les problèmes les algorithmes 1 et 2 sont équivalents;
  - il n'y a pas d'algorithme universellement meilleur qu'un autre;
  - il existe au moins un problème tel que l'aléatoire fasse de meilleurs résultats que un algorithme donné;

Exemple dans le cas d'attributs discrets ?

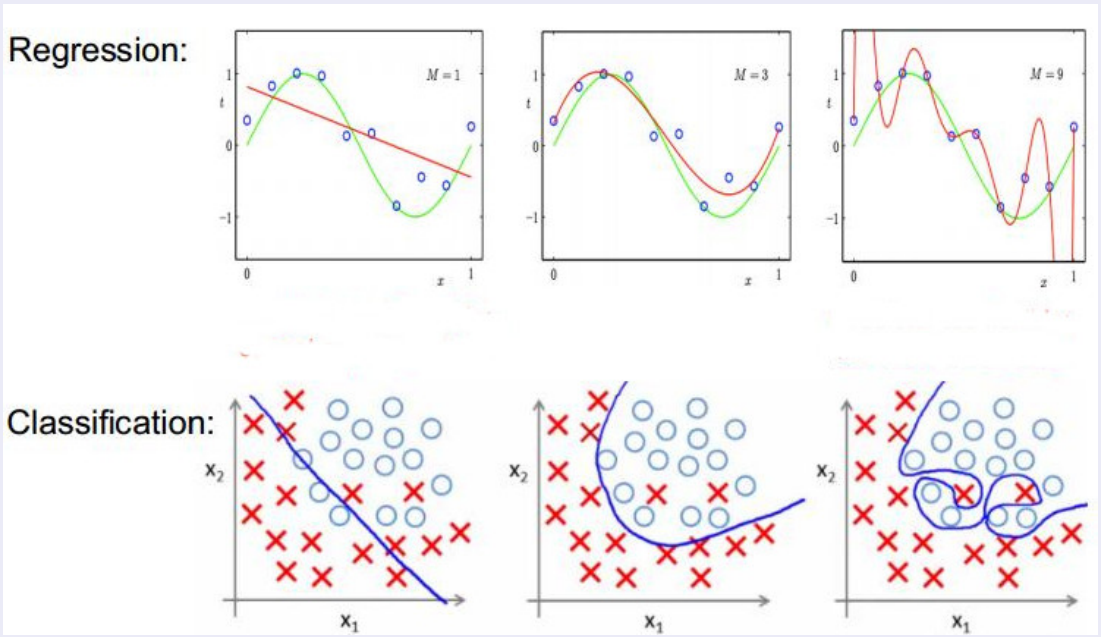
Risque empirique

Comment évaluer le risque ?

- Loi des grands nombres : $\hat{R}_n(f) \rightarrow_{n\rightarrow\infty} R(f)$
- Le risque empirique converge vers le risque bayésien

$\Rightarrow$ Choisir $f = min_f \hat{R}_n(f)$

Mais sur-apprentissage ...

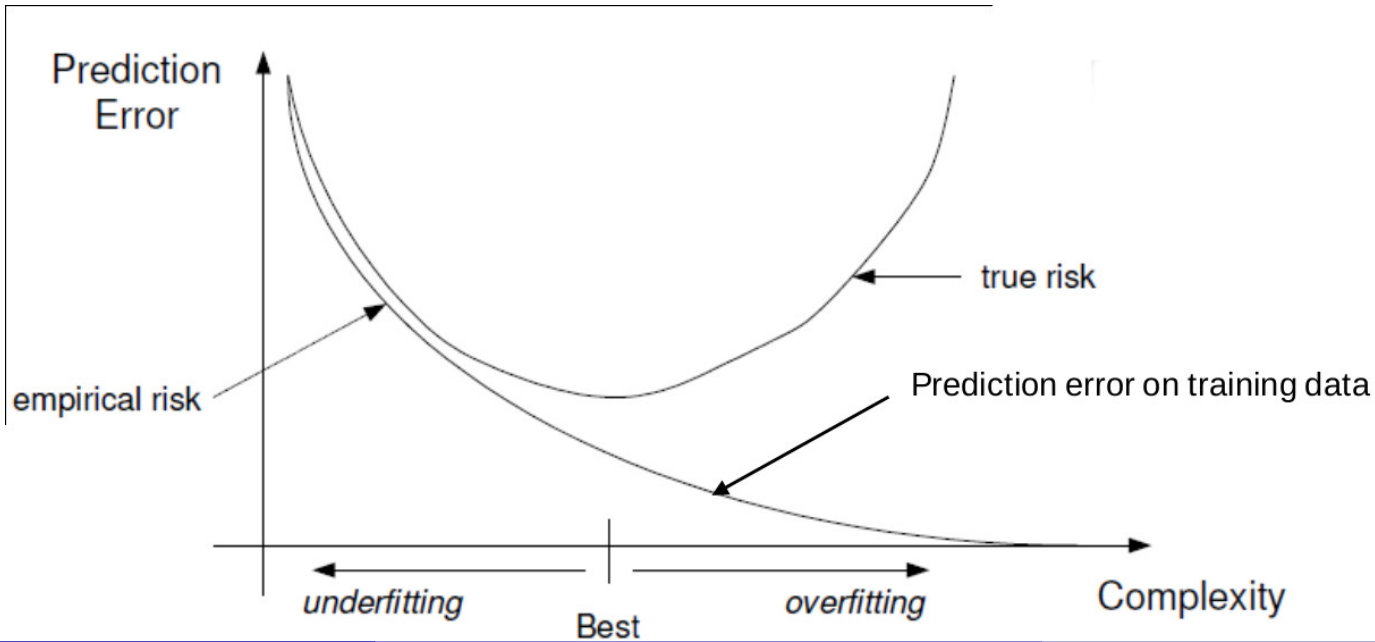


Solution : **restreindre** la famille de fonctions considérée : $f = inf_{f\in\mathcal{F}} \hat{R}_n(f)$

Minimisation structurelle du risque

Principe

- On ne considère que les fonctions dans une famille $\mathcal{F}$:
  $f = argmin_{f \in \mathcal{F}} \hat{R}_n(f) = argmin_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(y^i, f(\mathbf{x}^i))$
- vrai risque sur $\mathcal{F}$: $R^\star_{\mathcal{F}}$
- risque empirique sur $\mathcal{F}$: $\hat{R}^\star_{n,\mathcal{F}}$
- Condition nécessaire : $R^\star_{\mathcal{F}} - R^\star \geq 0$ doit être petit !
- Minimisation structurelle : faire évoluer $\mathcal{F}$ en fonction de n nombre d'exemples : $\mathcal{F}_{n+1} \supseteq \mathcal{F}_n$



Classifieur bayésien

Rappel

- Fonction de coût : 0-1 loss ($\ell(y,f(\mathbf{x})) = 1_{f(\mathbf{x})\neq y}$)
- $R^\star = min_f P(y \neq f(\mathbf{x}))$, $f^\star = argmin_f P(y \neq f(\mathbf{x}))$
- Propriétés :
  - $P(y \neq f^\star(\mathbf{x})) \leq P(y \neq f(\mathbf{x}))$ pour tout f
  - $f^\star = \begin{cases} 1 & \text{si } \eta(\mathbf{x}) > 1/2 \\ 0 & \text{si } \eta(\mathbf{x}) \leq 1/2 \end{cases}$ avec $\eta(\mathbf{x}) = \mathbb{E}[Y = 1|\mathbf{x}]$

Notations

Risque	Classifieur	Risque optimal
$R(f) = P(y \neq f(\mathbf{x}))$	$f^\star = argmin_f R(f)$	$R^\star = R(f^\star) = min_f R(f)$
	$f^\star_{\mathcal{F}} = argmin_{f\in\mathcal{F}} R(f)$	$R^\star_{\mathcal{F}} = R(f^\star_{\mathcal{F}}) = min_{f\in F} R(f)$
$\hat{R}(f) = \frac{1}{n}\sum_i 1_{f(\mathbf{x}^i)\neq y^i}$	$\hat{f}^\star_{n,\mathcal{F}} = argmin_{f\in\mathcal{F}} \hat{R}_n(f)$	$\hat{R}^\star_{n,\mathcal{F}} = \hat{R}(\hat{f}^\star_{n,\mathcal{F}}) = min_{f\in\mathcal{F}} \hat{R}_n(f)$

$\hat{f}^\star_{n,\mathcal{F}}$ est ce que produit l'algorithme d'apprentissage.

Compromis biais-variance

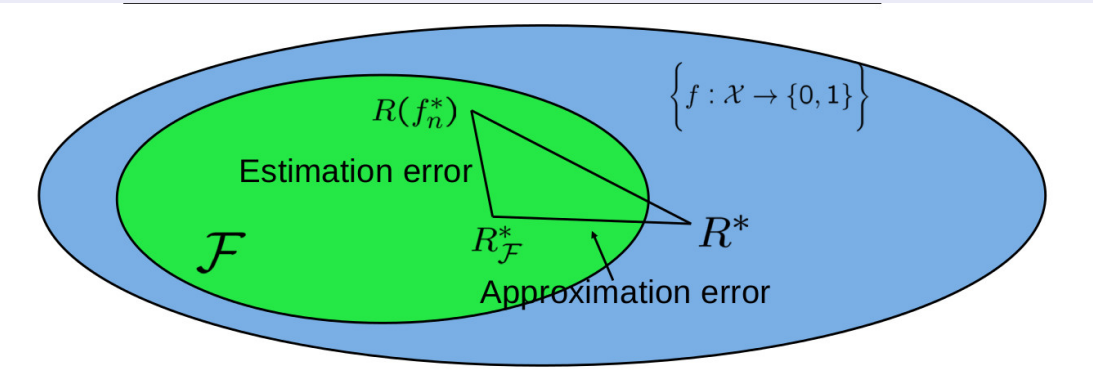
Décomposition de l'erreur

$$\hat{R}^\star_{n,\mathcal{F}} - R^* = \underbrace{\hat{R}^\star_{n,\mathcal{F}} - R^\star_{\mathcal{F}}}_{\text{erreur d'estimation}} \underbrace{+R^\star_{\mathcal{F}} - R^\star}_{\text{erreur d'approximation}}$$

$$\text{variance} \qquad\qquad \text{biais}$$

Biais : différence entre la meilleure fonction de $\mathcal{F}$ et le vrai risque

Variance : inhérent à l'échantillonnage des données



Sur/sous-apprentissage

- Si $\mathcal{F}$ trop grand : $R^\star_{\mathcal{F}} - R^\star$ petit, mais plus grande variance dans l'apprentissage de $\hat{f}^\star_{n,\mathcal{F}}$, sur-apprentissage
- Si $\mathcal{F}$ trop petit : $R^\star_{\mathcal{F}} - R^\star$ grand, erreur d'approximation grande, mais plus faible variance, sous-apprentissage

Un cas concret : espace de recherche fini

Contexte

- $\mathcal{F}$ est fini de k fonctions $\{f_1, \ldots, f_k\}$ et un jeu de données $\mathcal{D}$
- Soit $\hat{f}$ une fonction qui minimise l'erreur empirique
- Probabilité que la vrai erreur $e(\hat{f})$ de $\hat{f}$ soit plus grande que γ donné ?

Soit $f_i \in \mathcal{F}$ et $\{\mathbf{x}^j, y^j)\}_1^m \sim \mathcal{D}$ un échantillon iid

- Soit Z la variable aléatoire qui vaut 1 si $f_i(\mathbf{x}) \neq y$, $\mathbb{E}(Z) = e(f_i)$
- Soit Z_j la variable aléatoire qui vaut 1 si $f_i(\mathbf{x}^j) \neq y^j$, et $\hat{e}(f_i) = \frac{1}{m}\sum_1^m Z_j$.
- $\mathbb{E}(Z_j) = \mathbb{E}(Z) = e(f_i)$, donc d'après la borne d'Hoeffding : $P(|e(f_i) - \hat{e}(f_i)| \geq \gamma) \leq 2e^{-2\gamma^2 m}$

$\Rightarrow$ pour une fonction f_i donnée, l'erreur empirique est proche de l'erreur en généralisation.

Un cas concret : espace de recherche fini

Contexte

- $\mathcal{F}$ est fini de k fonctions $\{f_1, \ldots, f_k\}$ et un jeu de données $\mathcal{D}$
- Soit $\hat{f}$ une fonction qui minimise l'erreur empirique
- Probabilité que la vrai erreur $e(\hat{f})$ de $\hat{f}$ soit plus grande que γ donné ?

Soit A_i l'événement : $|e(f_i) - \hat{e}(f_i)| > \gamma$

- $P(A_i) \leq 2e^{-2\gamma^2 m}$
- $P(\exists f |\ |e(f) - \hat{e}(f)| > \gamma) = P(A_1 \cup \ldots \cup A_k) \leq \sum_{i=1}^{k} P(A_i) \leq 2ke^{-2\gamma^2 m}$
- $P(\neg\exists f |\ |e(f) - \hat{e}(f)| > \gamma) \geq 1 - \delta$, avec $\delta = 2ke^{-2\gamma^2 m}$
- Donne une vitesse de convergence : si $m \geq \frac{1}{2\gamma^2} log \frac{2k}{\delta}$, alors avec une proba de $1 - \delta$, la différence entre l'erreur empirique et celle de généralisation est plus petite que γ.
- Vitesse de convergence logarithmique par rapport aux nombres de fonctions.
- Remarque : $\gamma = \sqrt{\frac{1}{2m} log \frac{2k}{\delta}}$

Un cas concret : espace de recherche fini

Contexte

- $\mathcal{F}$ est fini de k fonctions $\{f_1, \ldots, f_k\}$ et un jeu de données $\mathcal{D}$
- Soit $\hat{f}$ une fonction qui minimise l'erreur empirique
- Probabilité que la vrai erreur $e(\hat{f})$ de $\hat{f}$ soit plus grande que γ donné ?

Soit $f^\star = argmin_{\mathcal{F}}\ e(f)$ la fonction optimale

Avec proba $1 - \delta$, pour un échantillon de m exemples,

- $e(\hat{f}) \leq \hat{e}(\hat{f}) + \gamma \leq \hat{e}(f^\star) + \gamma \leq e(f^\star) + 2\gamma$
- $e(\hat{f}) \leq (min_{f \in \mathcal{F}}\ e(f)) + 2\sqrt{\frac{1}{2m} log \frac{2|\mathcal{F}|}{\delta}}$
- Décomposition biais/variance : $\mathcal{F}$ large, alors 1er terme petit, le deuxième grand ...

Cas de famille infinie de fonctions

Question

- Combien de points en 1D un classifieur linéaire peut-il séparer ?
- En 2D ? Et un arbre de profondeur 2 ?

Définitions

- Un ensemble de points est *shattered* (pulvérisé) par un espace de fonction si pour tout partitionnement des points en deux ensembles il existe une fonction qui sépare les deux partitions.
- La VC-dimension (Vapnik-Chervonenkis) de $\mathcal{F}$ sur un espace de données $\mathcal{X}$ est la taille du plus grande ensemble fini de points de X pulvérisé par $\mathcal{F}$.
- Remarque : ce n'est pas *pour tout ensemble* de cardinal, mais *qu'il existe* un ensemble de cardinal.
- Fonctions linéaires : en dimension d, VC-dimension de $d+1$
- Borne PAC : $e(f) \leq e_D(f) + \sqrt{\frac{VC(\mathcal{F})}{m} log \frac{m}{VC(\mathcal{F})} + \frac{1}{m} log \frac{1}{\delta})}$
- Conséquence : le nombre d'exemples doit être linéaire en fonction de $VC(\mathcal{F})$.

Plan

Idée générale

Théorème du jury de Condorcet

soit un groupe de N voteurs indépendants qui choisissent correctement une réponse pour une question binaire avec une probabilité de $p > 0.5$. Alors quand N tend vers l'infini, la probabilité que l'aggrégation des votes soit la bonne réponse tend vers 1.

Conséquence

Combiner des classifieur faibles peut être meilleur que d'avoir un seul classifieur très expert . . .

- Bagging : (Breiman, 1996) faire varier l'ensemble d'apprentissage et apprendre des classifieurs indépendants
- Boosting (Shapire et Freund, 1990-1997) : classifieurs corrélés - chacun corrige l'erreur des autres - mais changement de la distribution du jeu d'apprentissage

Ensemble Learning

Principe

- Idée simple : considérer plusieurs (beaucoup) de classifieurs
- Avantage : réduit la variance si les classifieurs sont indépendants ! $Var(\hat{X}) = \frac{Var(X)}{n}$
- Mais qu'un jeu de données disponible...

$\Rightarrow$ Différentes techniques d'échantillonnage et d'agrégation pour varier les classifieurs appris

- Inférence : vote majoritaire pondéré sur l'ensemble des classifieurs.

Bagging

Bootstrap Aggregation — Breiman, 1994

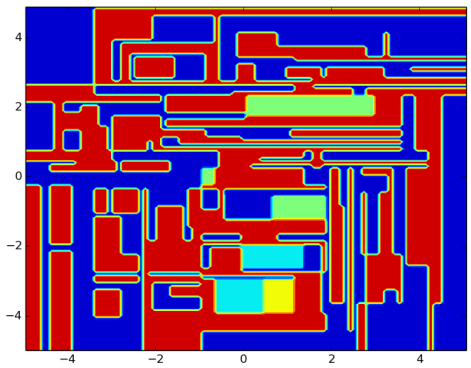
- constitution des N ensembles par tirage aléatoire **avec remise** d'un ensemble de même taille que l'original :
  $E \Rightarrow \{E_1, E_2, \ldots E_N\}$, avec $|E_i| = |E| = N$
- Apprendre $f_1, \ldots f_N$ sur ces ensembles d'apprentissage
- Classer $\mathbf{x}$ par moyennage ou vote de $f_1(\mathbf{x}), \ldots f_N(\mathbf{x})$
- Chaque donnée a une probabilité de $(1 - 1/n)^n$ d'être dans un E_i donné.

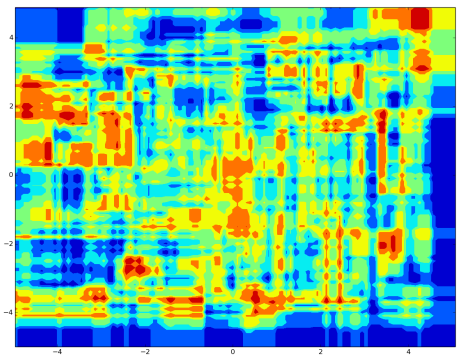
$\Rightarrow$ E_i contient en moyenne $1 - (1 - 1/n)^n\% = 63.2\%$ des instances initiales.

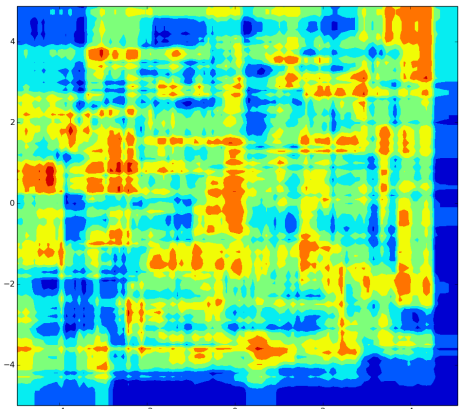
Un exemple : plusieurs arbres = une fôret

Principe

- A l'origine pour des considérations computationelles
- Deux facteurs d'aléa :
  - chaque arbre est appris sur un ensemble bootstrap de l'initial (bagging)
  - à chaque nœud, un sous-ensemble des dimensions est considéré uniquement, tiré aléatoirement.
- Décision au vote majoritaire (ou en moyenne pour la régression).
- Remarques : Effet de la profondeur ? Sur-apprentissage ?







Plan

Intuition

Classification de spam

Contrainte : utiliser que des règles atomiques sur les mots présents dans les emails

- Trouver le mot le plus fréquent parmi les spams et décider que tous ses emails contenant ce mot seront en spam
- Si l'email contient *buy* $\rightarrow$ score = -1
- mais certains emails contiennent *buy* sans que ce soit des spams $\Rightarrow$ corriger la règle
- Trouver le mot le plus fréquent parmi les non spams qui contiennent *buy* et donner un score de $+2$ à cette règle
- et ainsi de suite, corriger maintenant les spams qui contiennent ...

Intuition

Classification de spam

Contrainte : utiliser que des règles atomiques sur les mots présents dans les emails

- Trouver le mot le plus fréquent parmi les spams et décider que tous ses emails contenant ce mot seront en spam
- Si l'email contient *buy* $\rightarrow$ score = -1
- mais certains emails contiennent *buy* sans que ce soit des spams $\Rightarrow$ corriger la règle
- Trouver le mot le plus fréquent parmi les non spams qui contiennent *buy* et donner un score de $+2$ à cette règle
- et ainsi de suite, corriger maintenant les spams qui contiennent . . .

Intuition

Classification de spam

Contrainte : utiliser que des règles atomiques sur les mots présents dans les emails

- Trouver le mot le plus fréquent parmi les spams et décider que tous ses emails contenant ce mot seront en spam
- Si l'email contient *buy* $\rightarrow$ score = -1
- mais certains emails contiennent *buy* sans que ce soit des spams $\Rightarrow$ corriger la règle
- Trouver le mot le plus fréquent parmi les non spams qui contiennent *buy* et donner un score de $+2$ à cette règle
- et ainsi de suite, corriger maintenant les spams qui contiennent ...

Boosting

Principe

- Pourquoi faire de l'aléatoire quand on connait où on fait l'erreur ?
- Trouver plein de petits classifieurs approximativement bons sur de petites régions de l'espace.
- Idée : agréger plein de petits classifieurs, appris séquentiellement, chacun corrigeant les erreurs des précédents.

$\Rightarrow$ Besoin de classifieurs faibles ! (pourquoi ?)

Questions

- Qu'est ce qu'un classifieur faible ?

Boosting

Principe

- Pourquoi faire de l'aléatoire quand on connait où on fait l'erreur ?
- Trouver plein de petits classifieurs approximativement bons sur de petites régions de l'espace.
- Idée : agréger plein de petits classifieurs, appris séquentiellement, chacun corrigeant les erreurs des précédents.

$\Rightarrow$ Besoin de classifieurs faibles ! (pourquoi ?)

Questions

- Qu'est ce qu'un classifieur faible ?

$\Rightarrow$ classifieur peu expressif, arbres de faibles profondeurs, perceptrons ...

- Comment prendre en compte les erreurs ?

Boosting

Principe

- Pourquoi faire de l'aléatoire quand on connait où on fait l'erreur ?
- Trouver plein de petits classifieurs approximativement bons sur de petites régions de l'espace.
- Idée : agréger plein de petits classifieurs, appris séquentiellement, chacun corrigeant les erreurs des précédents.

$\Rightarrow$ Besoin de classifieurs faibles ! (pourquoi ?)

Questions

- Qu'est ce qu'un classifieur faible ?

$\Rightarrow$ classifieur peu expressif, arbres de faibles profondeurs, perceptrons . . .

- Comment prendre en compte les erreurs ?

$\Rightarrow$ Considérer une distribution des exemples w_t différente à chaque pas de temps

- Comment combiner les classifieurs ?

Boosting

Principe

- Pourquoi faire de l'aléatoire quand on connait où on fait l'erreur ?
- Trouver plein de petits classifieurs approximativement bons sur de petites régions de l'espace.
- Idée : agréger plein de petits classifieurs, appris séquentiellement, chacun corrigeant les erreurs des précédents.

$\Rightarrow$ Besoin de classifieurs faibles ! (pourquoi ?)

Questions

- Qu'est ce qu'un classifieur faible ?

$\Rightarrow$ classifieur peu expressif, arbres de faibles profondeurs, perceptrons . . .

- Comment prendre en compte les erreurs ?

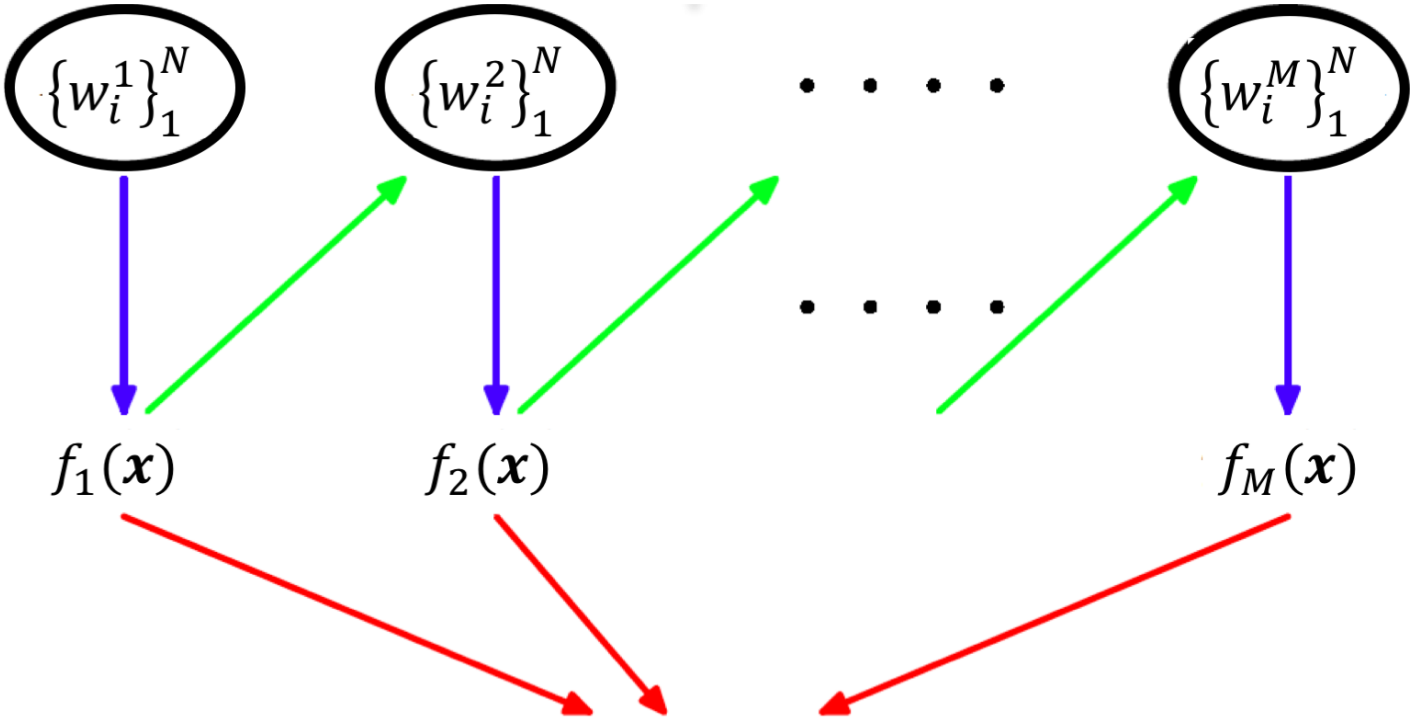
$\Rightarrow$ Considérer une distribution des exemples w_t différente à chaque pas de temps

- Comment combiner les classifieurs ?

$\Rightarrow$ Somme pondérée des classifieurs

- Combien de classifieurs apprendre ?

Boosting : formalisation



$$F_M(\boldsymbol{x}) = \text{sign}\left(\sum_m^M \alpha_m f_m(\boldsymbol{x})\right)$$

Chaque classifieur faible f_m est entrainé sur une pondération $\{\mathbf{w}_i^m\}$ des exemples.
Le classifieur final F_m est la somme pondérée par α_m des classifieurs f_m.

Boosting : déroulement

Initialisation

- $E = \{(x^i, y^i) \in \mathbb{R}^d \times \{-1, 1\}\}$ un ensemble de N données
- Distribution sur les données $\rightarrow$ un **poids** $w(i)$ sur chaque exemple i, avec la contrainte $\sum_{i=0}^{N} w(i) = 1$
- distribution uniforme au début : $w_0(i) = \frac{1}{N}$
- Pour un classifieur $f(\mathbf{x})$, l'erreur est :

$$\frac{1}{N} \sum_{i=0}^{N} w(i) \ell(f(\mathbf{x}^i), y^i)$$

- Définir une famille de classifieurs faibles $H = \{h : \mathbb{R}^d \rightarrow \{-1, +1\}\}$

AdaBoost

Principe

- $E = \{x^i, y^i\}$ un ensemble de données, distribution $w_t(i) = w_t^i$ sur ces données au temps t : $\sum_i w_t^i = 1$
- $\mathbf{h} = \{h_1, \cdots, h_T\}$ un ensemble de classifieurs,
- $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_T\}$ un ensemble de réels,
- $f_T(x) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) =< \boldsymbol{\alpha}, \mathbf{h} >$, $F_T(\mathbf{x}) = sign(f_T(\mathbf{x}))$ le classifieur pondéré.
- Objectif : trouver $(\mathbf{h}^\star, \boldsymbol{\alpha}^\star) = argmin_{\mathbf{h},\boldsymbol{\alpha}} \frac{1}{N} \sum_i 1_{F(\mathbf{x}^i) \neq y^i}$

Algorithme

1. Initialiser la distribution : $w_0(i) = \frac{1}{N}$
2. Apprendre h_t sur w_t
3. Calculer l'erreur $\epsilon_t = \sum_i w_t(i) 1_{h_t(\mathbf{x}^i) \neq y^i}$
4. Fixer $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
5. Mettre à jour $w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}^i)}$, avec $Z_t = \sum_i w_t(i) e^{-\alpha_t y^i h_t(\mathbf{x}^i)} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$

Remarques

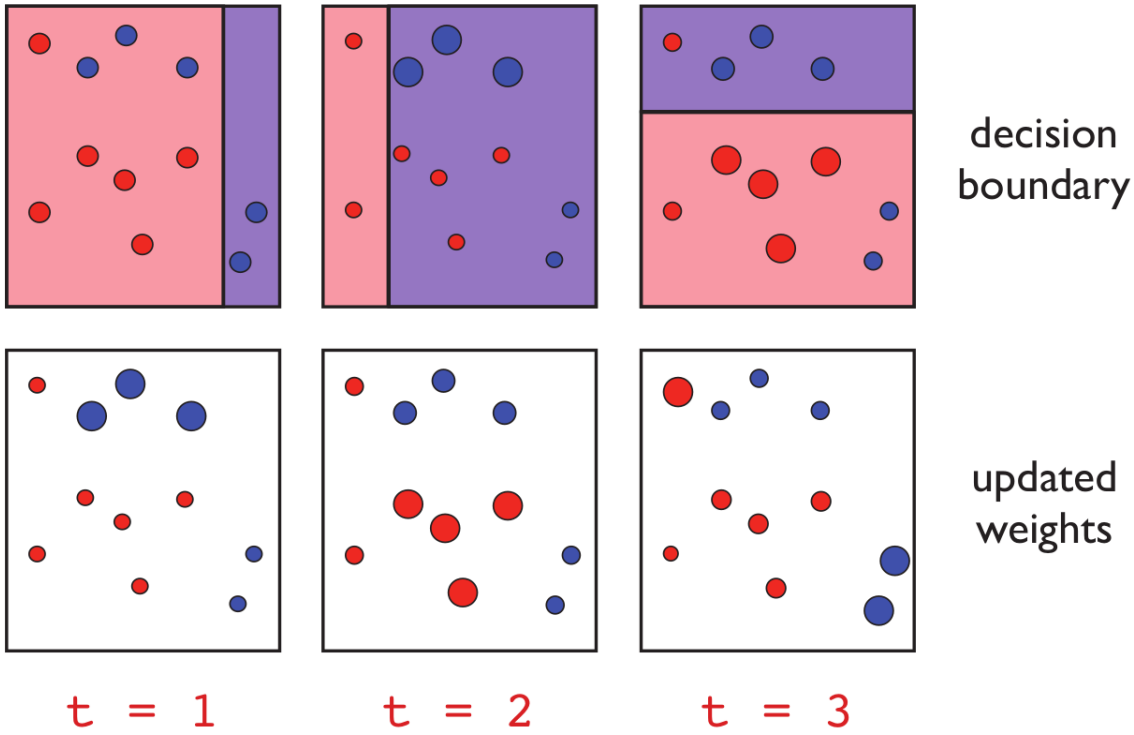
Considérations sur les poids

- $\epsilon_t < \frac{1}{2} \Rightarrow \alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) > 0$
- $\epsilon(h_a) < \epsilon(h_b) \Rightarrow \alpha_a > \alpha_b$
- $e^{-y\alpha_t h_t(\mathbf{x})} = \begin{cases} e^{-\alpha_t} < 1 & \text{si } h_t(\mathbf{x}) = y \\ e^{\alpha_t} > 1 & \text{si } h_t(\mathbf{x}) \neq y \end{cases}$

Considérations sur la distribution

- $w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{-\alpha_t y^i h_t(\mathbf{x}^i)} = \frac{1}{Z_t Z_{t-1}} w_{t-1}(i) e^{-y^i(\alpha_t h_t(\mathbf{x}^i) + \alpha_{t-1} h_{t-1}(\mathbf{x}^i))}$
  $\cdots = \frac{1}{Z_t \cdots Z_1} w_1(i) e^{-y^i(\alpha_t h_t(\mathbf{x}^i) + \cdots + \alpha_1 h_1(\mathbf{x}^i))}$
- On montre que $Z = Z_1 \cdots Z_t = \frac{1}{N} \sum_{i=1}^{N} e^{-y^i f_t(\mathbf{x}^i)}$
- Et que $Err(F) \leq Z$

Exemple



(a)

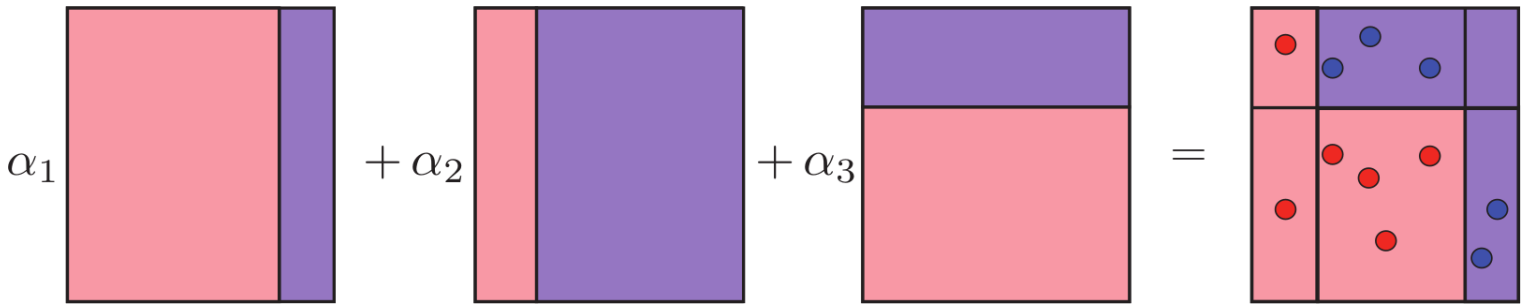


Image de Rostamizadeh et. al. 2012

Exemple

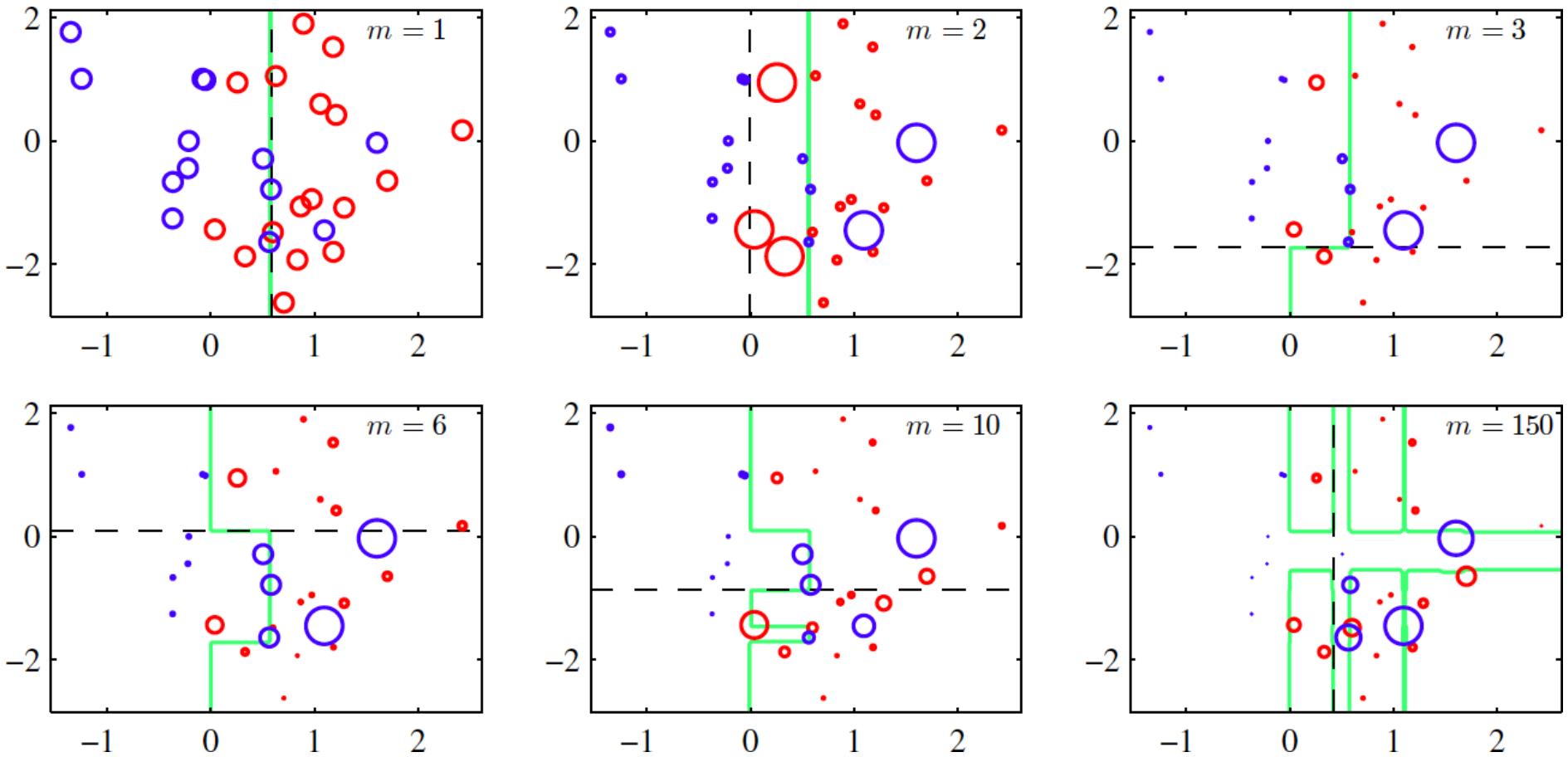
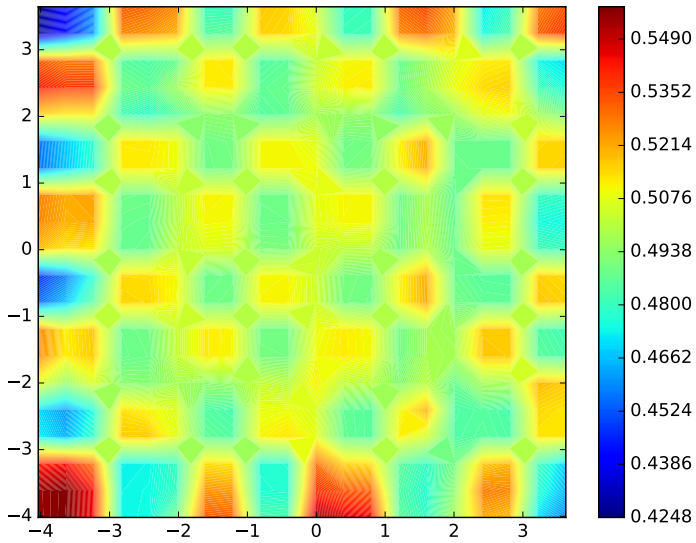
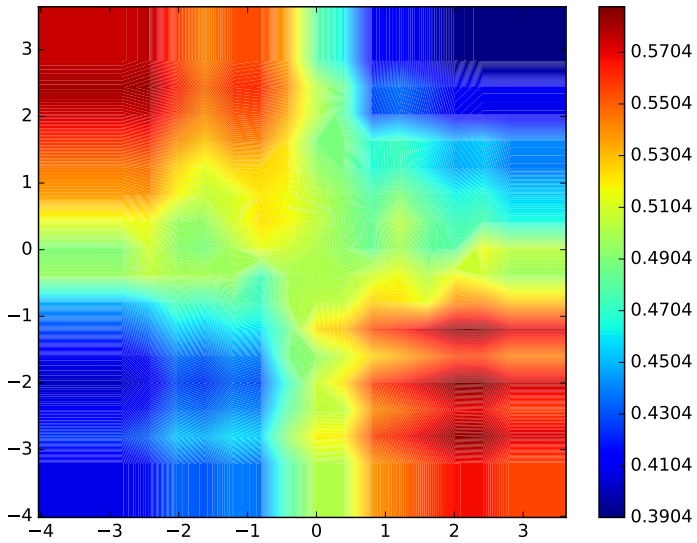


Image de Bishop (2006)

Illustrations





Boosting généralisé : Gradient Boosting

Formalisation

- On cherche $F_T = \sum_{i=1}^{T} \alpha_t h_t(\mathbf{x})$
- De manière gloutonne :

  $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \text{argmin}_{h_t} \left[\sum_{i=1}^{N} L(y^i, F_{t-1}(\mathbf{x}^i) + h_t(\mathbf{x}^i)) \right]$
- Problème : optimisation impossible en général

$\Rightarrow$ Descente de gradient : $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) - \gamma \sum_{i=1}^{N} \nabla_{F_{t-1}} L(y^i, F_{t-1}(\mathbf{x}^i))$

et choisir $\gamma_t = \text{argmin}_{\gamma} \sum_{i=1}^{N} L(y^i, F_{t-1}(\mathbf{x}^i) - \gamma \nabla_{F_{t-1}} L(y^i, F_{t-1}(\mathbf{x}^i)))$

Dans le cas de la MSE

- $\nabla_{F_{t-1}} L(y^i, F_{t-1}(\mathbf{x}^i)) = K(y^i - F_{t-1}(\mathbf{x}^i))$ le résidu en chaque exemple i
- Trouver une fonction h_t qui approxime au mieux le résidu
- Trouver le α_t qui minimise $\sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}^i) + \alpha_t h_t(\mathbf{x}^i))$

Conclusions

Sur le bagging

- Très utilisé ! (kinect, les gagnants de netflix)
- Facile à mettre en place, peut traiter de grosses masses de données (parallélisation), en apprentissage et en inférence

Boosting

- Classifieurs faibles : Stump (arbre à un niveau), naive bayes, perceptron,. . .
- Adaptable sous beaucoup d'autres formes (gradient tree boosting, gradient boosting)
- Adapté au très grande masse de données et données sparse (ciblage publicitaire par exemple)

Plan

Mesures d'évaluation

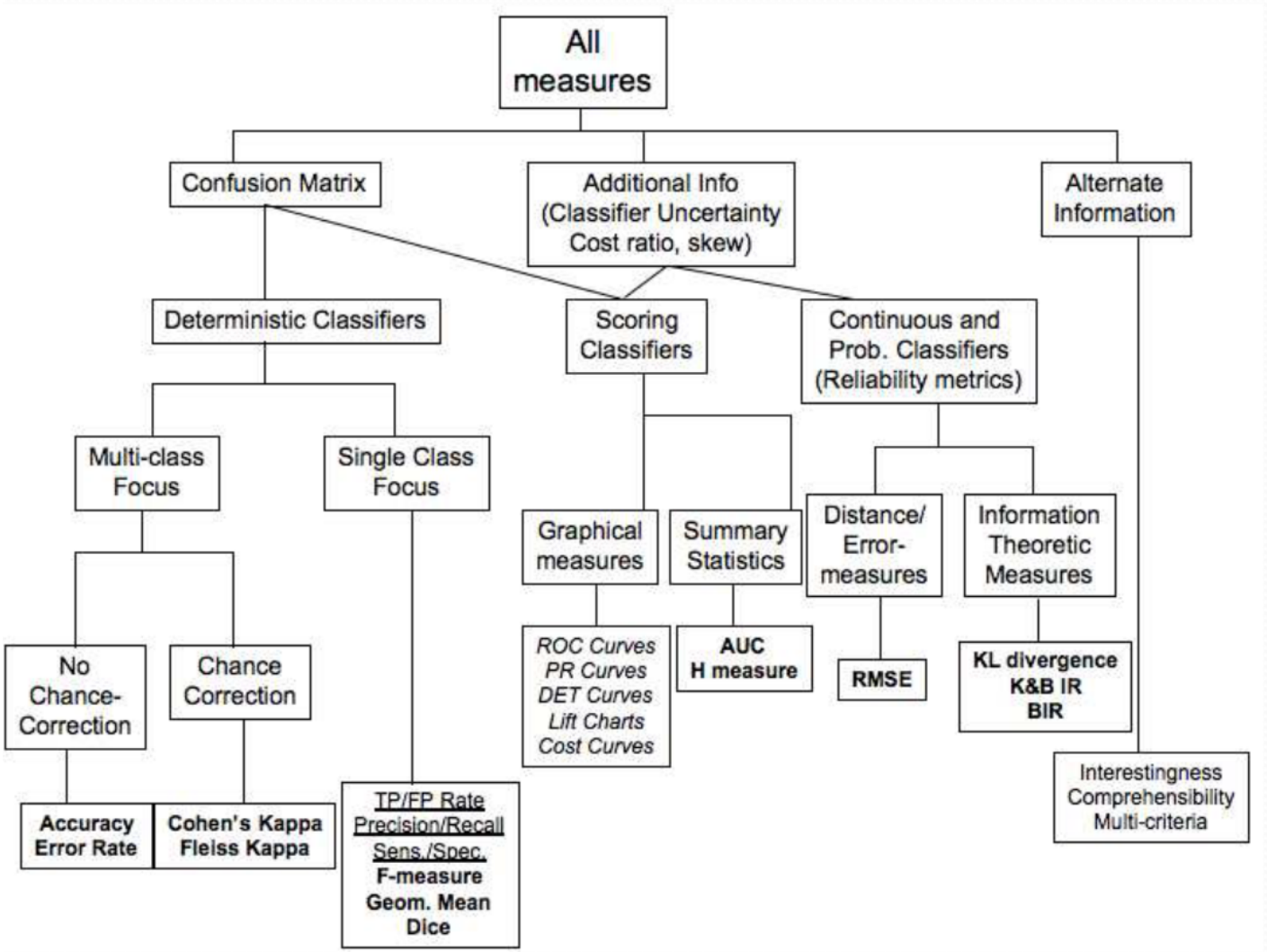
Objectifs

- Estimer la qualité des prédictions fournies par une approche
- Comparer des approches entre elles sur un problème donné
- Comparer des algorithmes sur un ensemble de problèmes

Le résultat dépend

- Choix de la mesure
- Choix du protocole de test (paramétrisation)
- Choix de l'échantillage

Une mesure unique ?



Tutorial icmla 2011, N. Japkowicz

Matrice de confusion

Contexte

- Un problème de classification binaire, étiquettes positif/négatif
- TP : Vrai positif (*True positive*), TN : Vrai négatif (*True negative*)
- FP : Faux positif (*False positive*), FN : Faux négatif (*False negative*)

Matrice de confusion

	Label $+$	Label $-$
$f(x) = +1$	TP	FP
$f(x) = -1$	FN	TN
	$P = TP + FN$	$N = FP + TN$

Mesures dérivées

- Erreur $0-1$: $\frac{FP+FN}{P+N}$
- Précision : $\frac{TP}{TP+FP}$
- Rappel (TP rate) : $\frac{TP}{P}$
- FP Rate : $\frac{FP}{N}$
- $F_\beta = (1+\beta^2)\frac{\text{precision}\times\text{rappel}}{\beta^2\text{precision}+\text{rappel}}$

Exemple (ou le problème du déséquilibre)

	Label +	Label −
$f(x) = +1$	200	100
$f(x) = -1$	300	400
	500	500

- Erreur : 40%
- Précision : 66%, Rappel : 40%
- F_1 : 0.5

	Label +	Label −
$f(x) = +1$	400	300
$f(x) = -1$	100	200
	500	500

- Erreur : 40%
- Précision : 57%, Rappel : 80%
- F_1 : 0.66

	Label +	Label −
$f(x) = +1$	200	100
$f(x) = -1$	300	400
	500	500

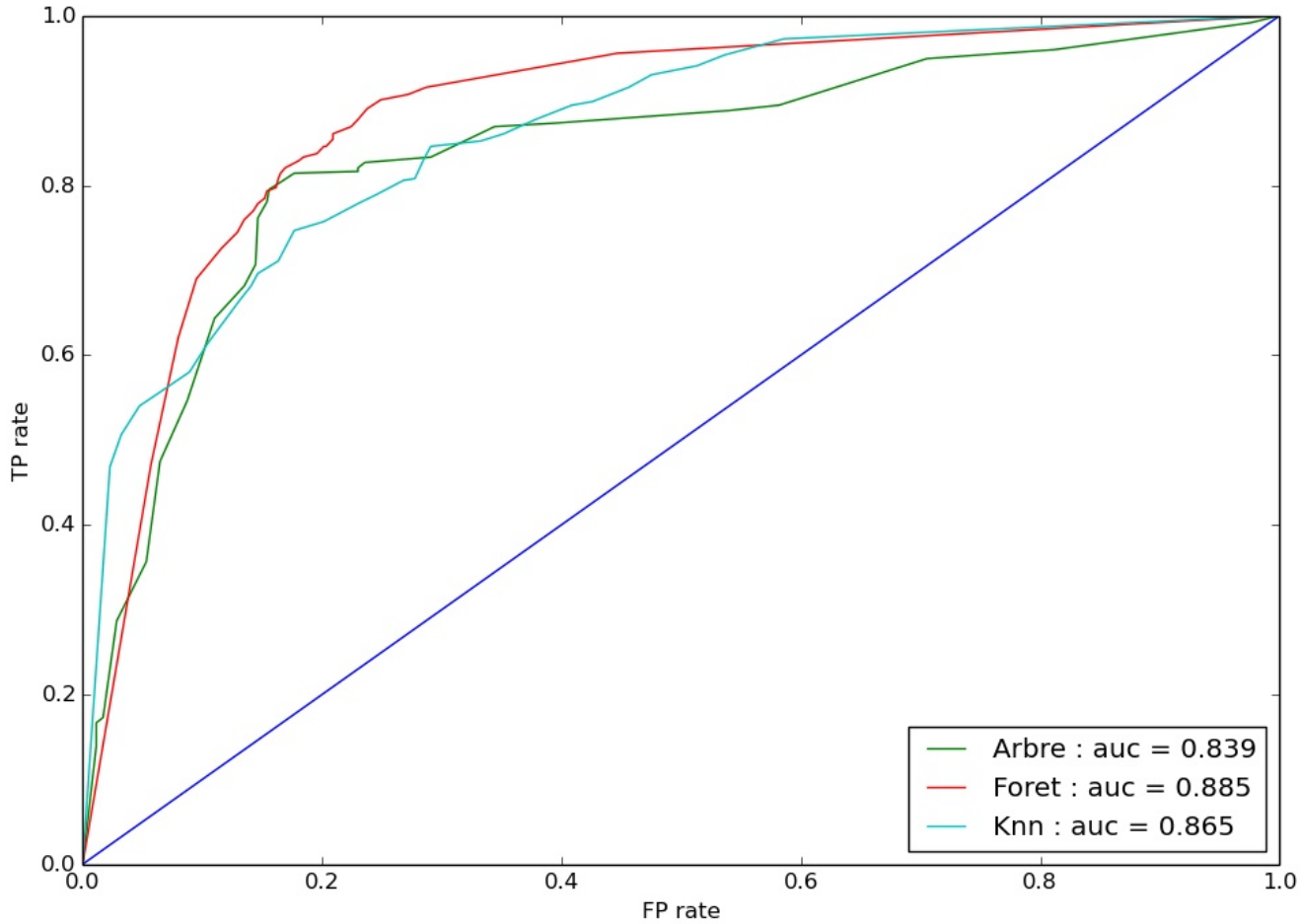
- Erreur : 40%
- Précision : 66%, Rappel : 40%
- F_1 : 0.5

	Label +	Label −
$f(x) = +1$	200	100
$f(x) = -1$	300	0
	500	100

- Erreur : 66%
- Précision : 66%, Rappel : 40%
- F_1 : 0.5

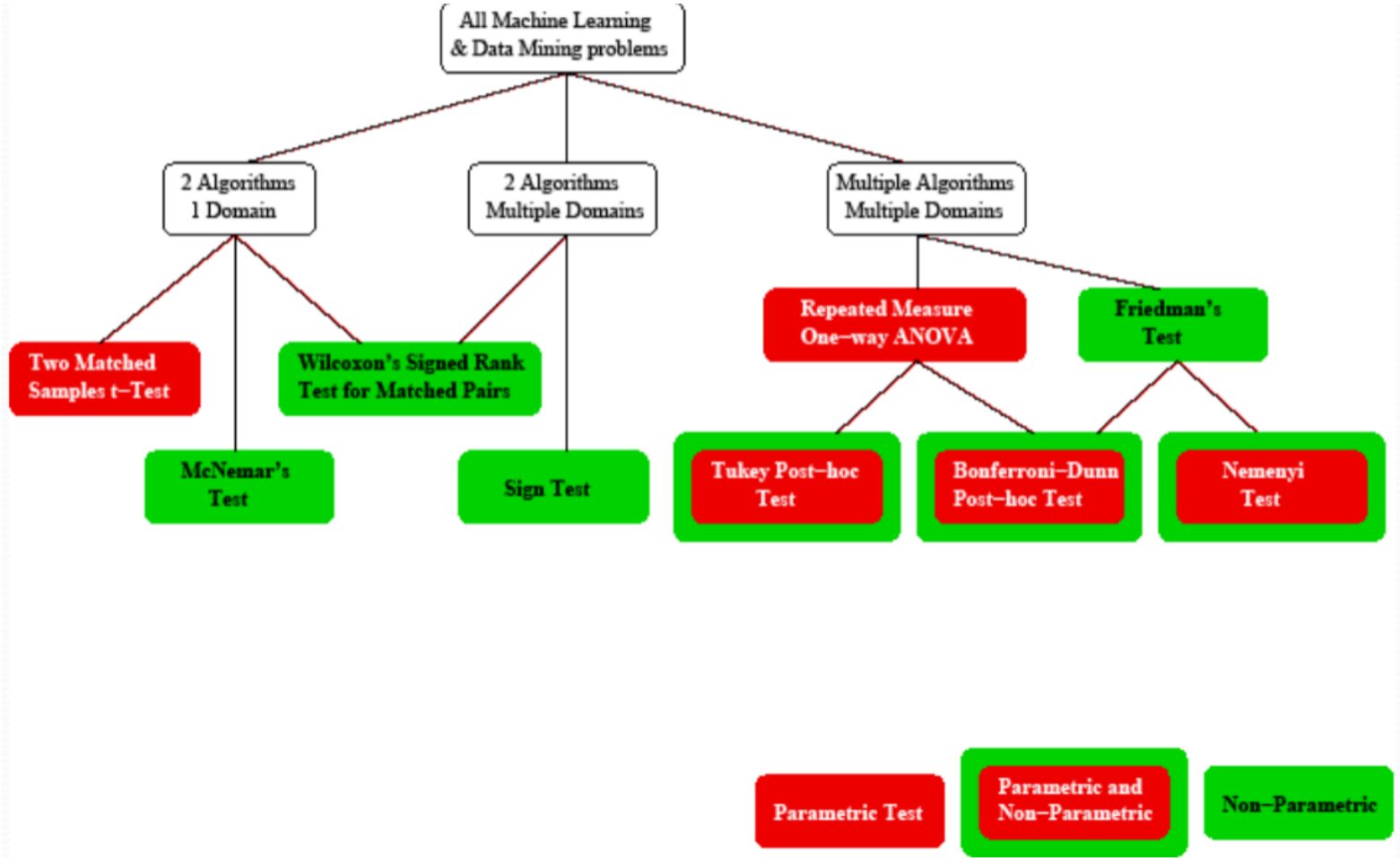
Courbe ROC et AUC

- Courbe ROC : TP rate en fonction du FP rate
- permet de calibrer un classifieur
- mesure d'intérêt : AUC, aire sous la courbe



Comment comparer deux algos ?

- Test statistique



Plan

Cas usuel

Contexte

- Classes : $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$
- Classification binaire ne marche pas directement

Approches "naïves" utilisant la classification binaire

- One-versus-one : matrice $M_{ij} = C_i$ vs C_j
- One-versus-all : vecteur $M_i = C_i$ vs $\{C_{j \neq i}\}$

Adaptation de la classification binaire

- Arbres, forets, k-nn : adaptation triviale
- SVMs multi-classes
- Réseau de neuronnes : vecteur de sortie $\mathbf{y}$ et softmax : $p(y_j) = \frac{e^{y_j}}{\sum_i e^{y_i}}$

Très grand nombre de classes

Problèmes des approches usuelles

- Coût d'une classification τ
- au mieux linéaire en fonction de K : temps τK
- grand nombre de dimensions

$\Rightarrow$ passage à l'échelle difficile en temps de calcul et en perfs

Deux grandes familles d'approche

- Approche *flat* : plonger les classes dans un espace $\mathbb{R}^{K'}$, $K' << K$
  Intérêt : $K'\tau$ pour trouver la bonne classe
- Approche *hiérarchique* : organiser les classes hiérarchiquement dans un arbre de classes
  Intérêt : inférence en $log(K)\tau$ pour un arbre binaire

Approches Error Correcting Output Code (ECOC)

Principe

- Plonger les classes dans $\mathbb{R}^{K'}$, $K' << K$
- Codage : une classe $\Leftrightarrow$ un code dans K'
- Inférence $=$ codage : $f : X \rightarrow K'$, $f(x)$ donne un code dans K'
- Décodage : classe dont le code est le plus proche

En pratique

- Un code c^i : un vecteur ternaire de K : $(-1, 0, 1, \ldots, 0, 1)$
- A chaque code, un classifieur binaire f_i qui sépare $\{C_j | c^i_j > 0\}$ et $\{C_j | c^i_j < 0\}$
- Matrice M de codage de K' : matrice $K' \times K$ des $M_{ij} = c^i_j$
- Codage d'une classe C_j : $(c^1_j, c^2_j, \ldots c^{K'}_j)$
- codage d'un exemple : $(f_1(x), f_2(x), \ldots, f_{k'}(x))$
- Inférence : $argmin_j d(f(x), M_{.j})$ en $O(K'\tau + K)$

Approche hiérarchique

Objectif

- Construire un arbre de partitionnement (hard ou soft) des classes
- Pour un nœud n :
  - un ensemble $\mathcal{C}_n$ de classes, pour les fils $n_1, \ldots, n_c$ sous-ensembles $\mathcal{C}'_{n_1}, \ldots, \mathcal{C}'_{n_c} \subset \mathcal{C}_n$, et $\bigcup \mathcal{C}'_{n_j} = \mathcal{C}_n$
  - un classifieur f_n à valeur dans $\{n_1, \ldots, n_c\}$
- Racine : ensemble de toutes les classes, feuilles : une seule classe
- Classification : un chemin dans l'arbre (en utilisant f_n), classe de la feuille

Problématiques

- Construire l'hiérarchie :
  - information a priori sur les classes : ontologie ou hiérarchie des classes
  - apprentissage de l'hiérarchie : clustering, approches gloutonnes
- Apprendre les classifieurs : problème de données non équilibrés
- Correction des erreurs : redondance des classes dans les nœuds de l'arbre