Ensemble learning
Mesures d'évaluation
Multi-classe

Cours 6
ARF Master DAC

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr
http://webia.lip6.fr/~baskiotisn

équipe MLIA, Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université - Université Pierre et Marie Curie (UPMC)

S2 (2017-2018)

Ensemble Learning

Principe

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

$\Rightarrow$ Différentes techniques d'échantillonage et d'aggrégation pour varier les classifieurs appris

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

Plan

Bagging

Bootstrap Aggregation — Breiman, 1994

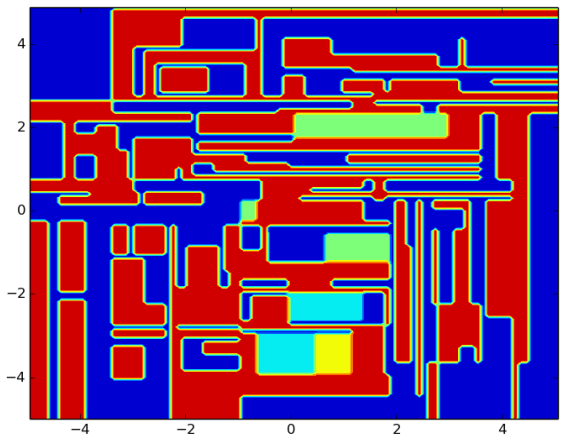
- constitution des N ensembles par tirage aléatoire **avec remise** d'un ensemble de même taille que l'original :
  $E \Rightarrow \{E_1, E_2, \ldots E_N\}$, avec $|E_i| = |E| = N$
- Apprendre $f_1, \ldots f_N$ sur ces ensembles d'apprentissage
- Classer $\mathbf{x}$ par moyennage ou vote de $f_1(\mathbf{x}),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

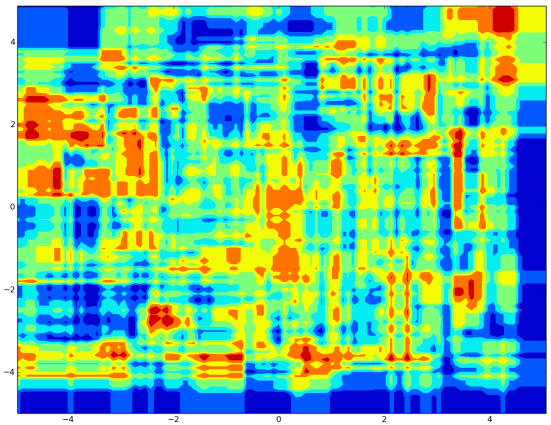
$\Rightarrow$ E_i contient en moyenne $1 - (1 - 1/n)^n\% = 63.2\%$ des instances initiales.

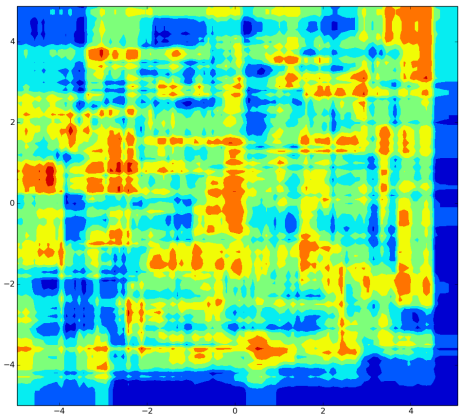
Un exemple : plusieurs arbres = une fôret

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 : 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 $w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}^i)}$

Remarques

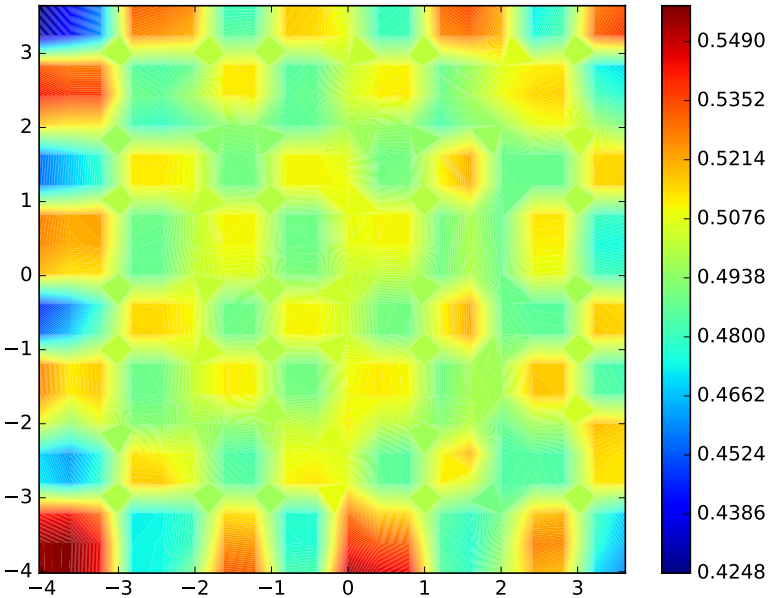
Considérations sur les poids

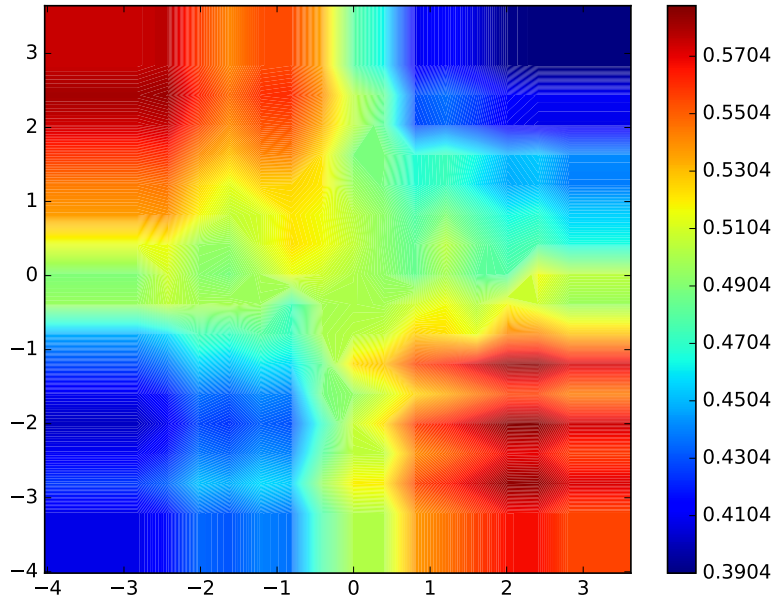
- $\epsilon_t < \frac{1}{2} \Rightarrow \alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) > 0$
- $\epsilon(h_a) < \epsilon(h_b) \Rightarrow \alpha_a > \alpha_b$
- $e^{-y\alpha_t h_t(\mathbf{x})} = \begin{cases} e^{-\alpha_t} < 1 & \text{si } h_t(\mathbf{x}) = y \\ e^{\alpha_t} > 1 & \text{si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

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 $Err(F) \leq Z$

Illustrations





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

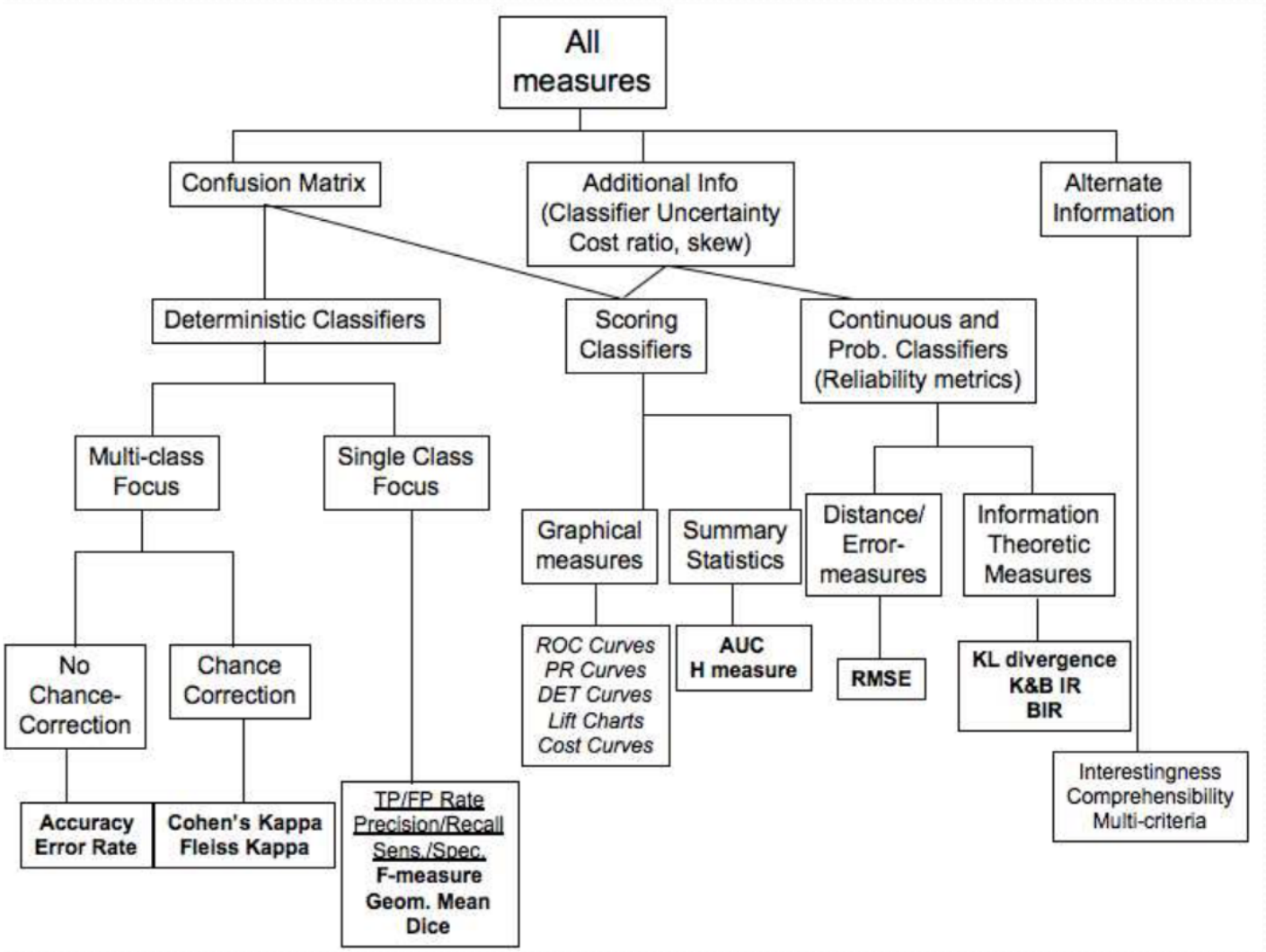
Objectifs

- 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 : 60%
- Précision : 40%, Rappel : 40%
- F_1 : 0.4

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

- Erreur : 60%
- Précision : 66%, Rappel : 80%
- F_1 : 0.66

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

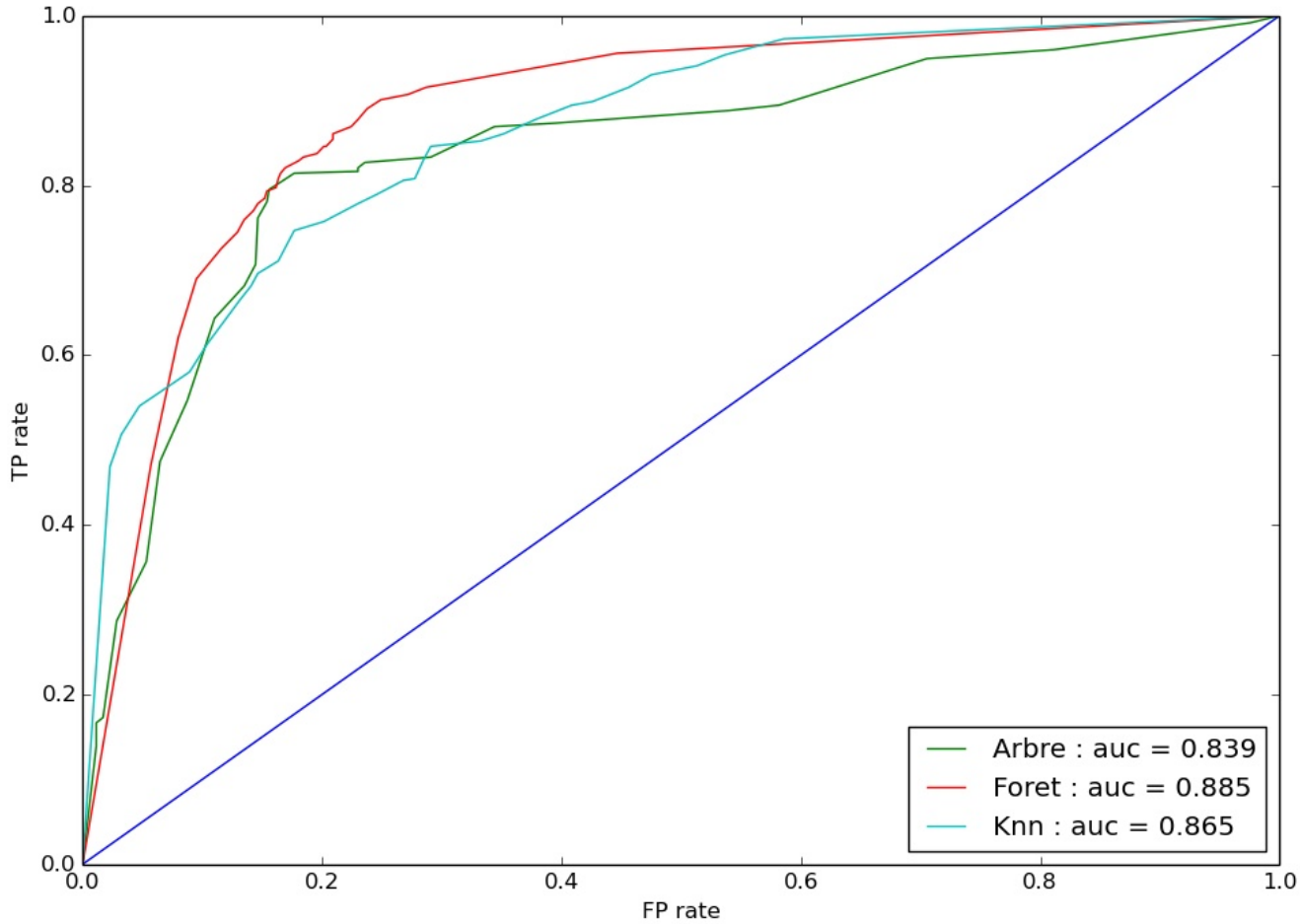
- Erreur : 60%
- Précision : 66%, Rappel : 40%
- F_1 : 0.5

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

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

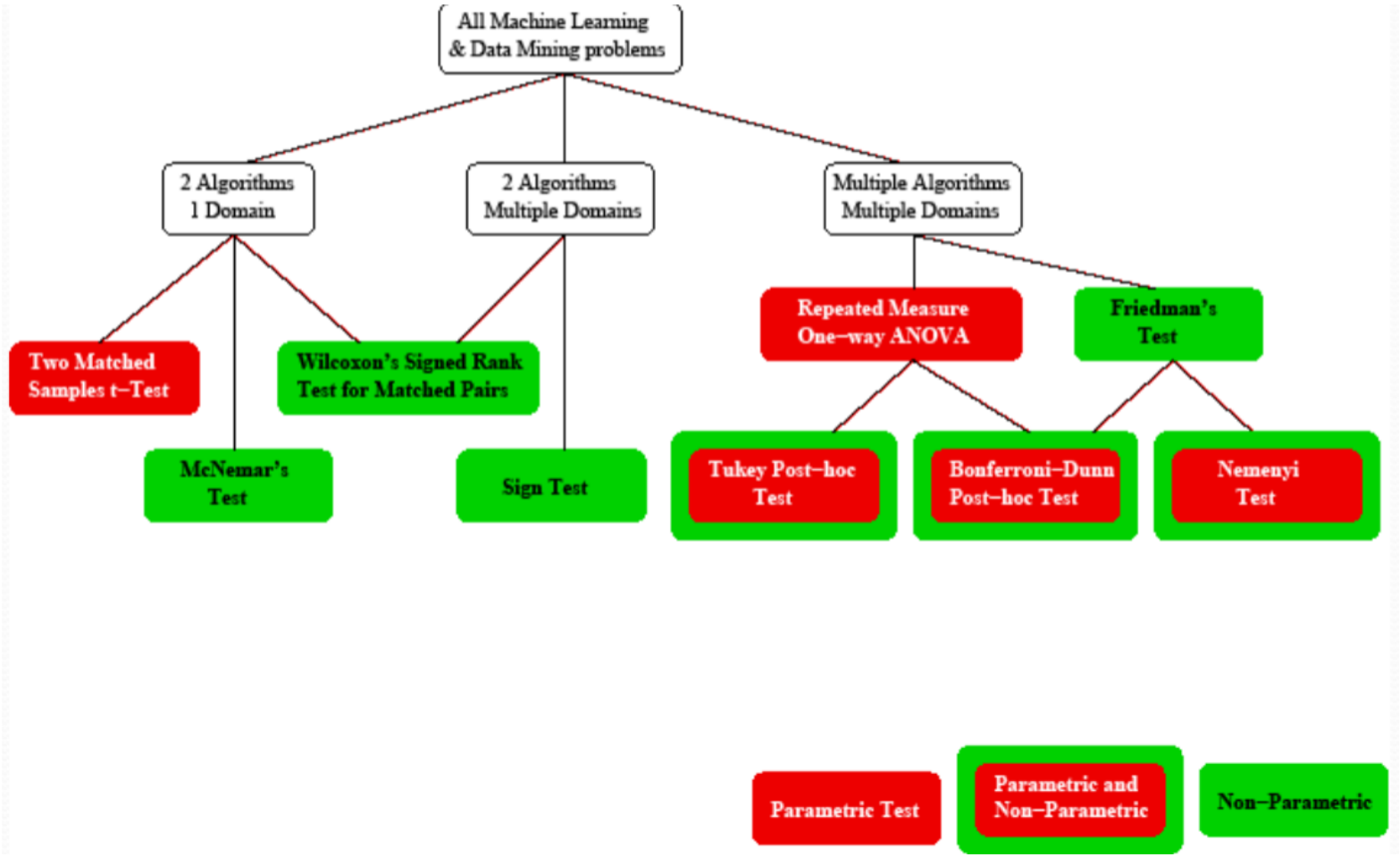
Courbe ROC et AUC

- 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