

Big Data Mining

Cas d'entreprise

Master 2 BIBD et Master 2 SISE

Guillaume Metzler

guillaume.metzler@univ-lyon2.fr



**INSTITUT
de la
communication**



Université de Lyon, Lyon 2, ERIC EA3083, Lyon, France

Automne 2020

Introduction

A propos

- PME en Informatique de 10 salariés environ
- Basée à Villefontaine
- Essentiellement des ingénieurs et un chercheur



A propos

Activités principales

Buy now.
Pay in monthly
installments.



Facilité de paiement



Fluidité de passage



Sécurisation transactions
chèques

Autres activités

- Assistance dans la gestion de PV
- Assistance dans la gestion de personnels

→ **Une thèse centrée sur la détection de fraudes par chèques.**

A propos

Fraude par chèque ?



- Impayé (solde non disponible sur le compte)
- Faux chèque

Identité non réelle

Série de caractères incorrects dans la ligne CMC7

A propos

Fraude par chèque ?



- Impayé (solde non disponible sur le compte)
- Faux chèque

Identité non réelle

Série de caractères incorrects dans la ligne CMC7

Quelques statistiques :

10 mois de transactions
(20/03/2016 to 21/10/2016)

- environ 3.2 millions de transactions
- pour 195 millions d'euros
- 20 000 fraudes ou impayées (0.6%)
- représentent 2 millions d'euros (1.1%)

Apprentissage déséquilibré et Big Data

A propos

Les tâches quotidiennes de l'entreprise, valables pour d'autres entreprises

- gérer l'intégration continue des données avec les clients
- gérer la base de données (période + endroit de stockage)
- gérer un flux **massif** de données permanent
- apprendre à détecter des fraudes
- avoir des modèles qui **répondent rapidement (20 ms environ)** mais s'apprennent aussi en un temps "raisonnable"

Gestion de la masse de données

On va ici s'affranchir de ce qui est relatif à la gestion de la base de données ... cela est relatif à un autre cours. En revanche, vous pouvez faire de la gestion de base de données avec  si vous avez un serveur SQL.

```
1 # Necessaire pour se connecter a un serveur SQL
2 library(RODBC)
3
4 # D'autres librairies qui peuvent etre utiles dans ce contexte
5 library(sqldf)
6 library(tcltk)
7 library(rJava)
8 library(filehash)
9
10 # creer une base de donnees vide
11 dbCreate("my_data_base")
12 db <- dbInit("my_data_base")
13
14 # se connecter a un serveur SQL, requete et execution
15 LibSQL <- odbcConnect( dsn = "nom_base", uid = "...", pwd = "...")
16 req = paste( [On ecrit sa requete SQL classiquement])
17 sqlQuery(LibSQL,req)
18 #ou
19 dbInsert(db, "my_db", sqlQuery(LibSQL,req))
20
21 # Fermer la connexion
22 odbcClose(LibSQL)
```

Exemple de code

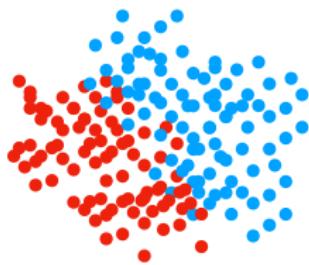
Voilà un petit exemple issu de ma thèse

```
1 library(RODBC)
2 library(lubridate)
3 dbCreate("mydbh")
4 dbh <- dbInit("mydbh")
5 LibSQL2=odbcConnect(dsn="Casino",uid="gmetzler",pwd="W5Y@8c+v")
6
7 # La requete ci-dessous consiste a recuperer des transactions anterieures a chaque
     transaction identifiees par une ZIBZIN (numero non unique donc la data va jouer un
     role primordial)
8
9 req=paste("SELECT    'A'+ZIBZIN as ZIBZINPR , DateTransaction as DateTransactionPR ,
10           NumCheque as NumChequePR, Montant as MontantPR, IDMagasin,
11           ISNULL(CAST(CAST(CONVERT(CHAR(10), DateTransaction, 112) AS DATETIME)
12                     AS Float),0) AS CumulDate1,
13           ScoringFP1 as ScoringFP1PR, ScoringFP2 as ScoringFP2PR ,
14           ScoringFP3 as ScoringFP3PR,
15           CONVERT(DATETIME, DateTransaction , 102) as DateJPR,
16           DATEDIFF(day,'",origin,"',DateTransaction) as NbJPR
17           FROM          Casino.dbo.EncaissementGlissant
18           WHERE        (EncaissementGlissant.DateTransaction >= CONVERT(DATETIME , '",
19           Datedebutapprentissage,"', 102) - ",seuilnbjourhistorique,") AND (
20           EncaissementGlissant.DateTransaction < CONVERT(DATETIME , '",
21           Datefinapprentissage,"', 102)+1)")

# L'historique des transactions est ensuite stockee dans la base de donnees "dbh"
22
23 dbInsert(dbh,"histo",sqlQuery(LibSQL2,req))
24 odbcClose(LibSql2)
```

Apprentissage dans un contexte déséquilibré

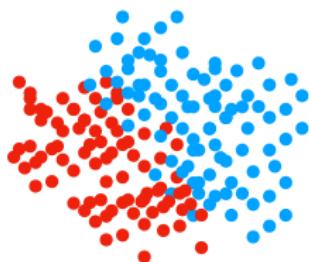
Balanced dataset



Positives \simeq Negatives

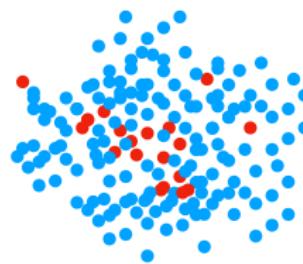
Apprentissage dans un contexte déséquilibré

Balanced dataset



Positives \simeq Negatives

Imbalanced dataset



Positives \ll Negatives

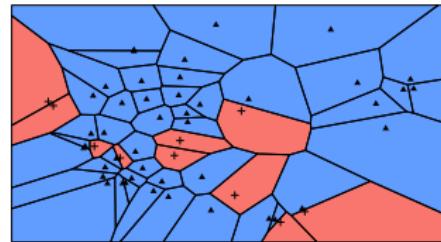
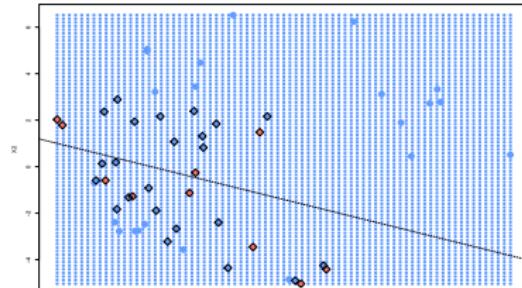
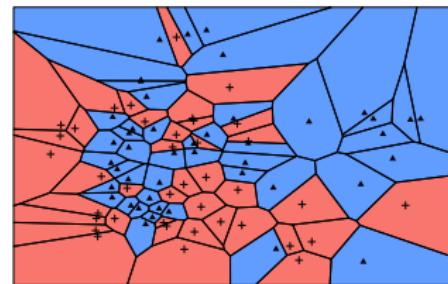
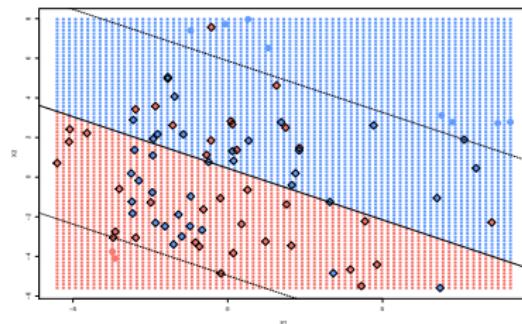
On minimise une surrogate de $\frac{1}{m} \sum_{i=1}^m 1_{\{\hat{y}_i \neq y_i\}}$ ce qui nous conduit à

focus les deux classes

focus classe majoritaire

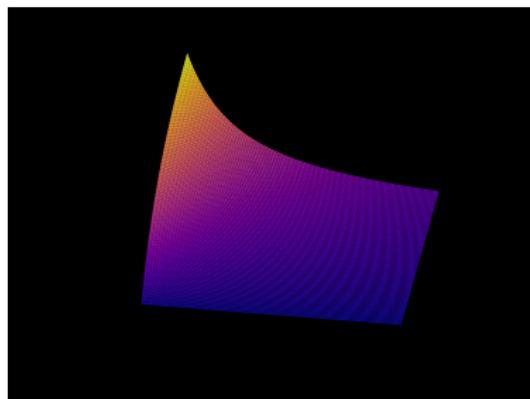
Apprentissage dans un contexte déséquilibré

Exemple d'un SVM linéaire et d'un k -NN avec 50% et 20% de données positives.



Apprentissage dans un contexte déséquilibré

Usage de mesures appropriées

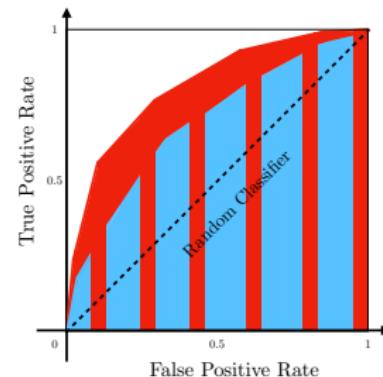


$$F_\beta = \frac{(1 + \beta^2)(P - FN)}{(1 + \beta^2)P - FN + FP}$$

G-mean

Mean Average Precision

Recall



$$\mathbb{P}[f(x_+) > f(x_-)]$$

Precision

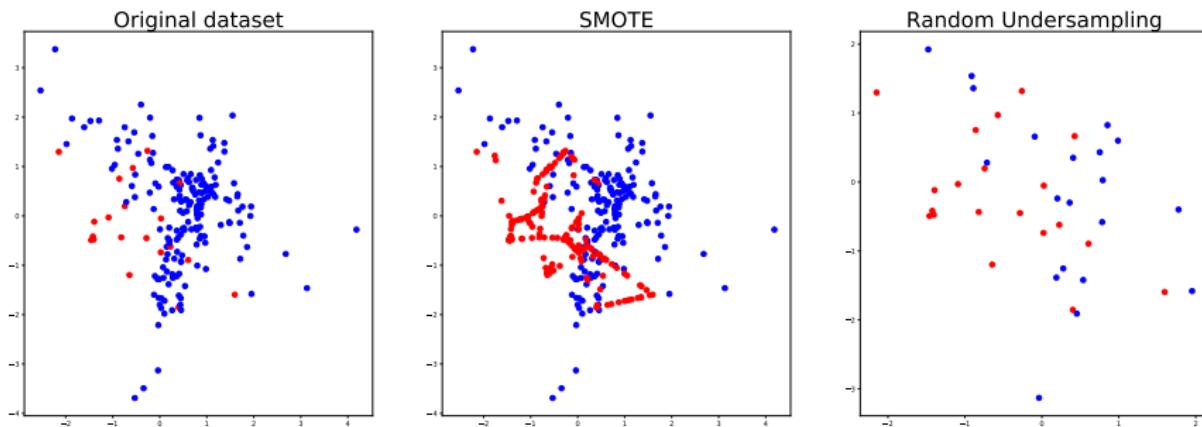
Average Precision

MCC

False Positive Rate

Apprentissage dans un contexte déséquilibré

Utiliser des stratégies d'échantillonnages



- Oversampling: Random - SMOTE - BorderSMOTE, ...
- Undersampling: Random - Tomek Link - ENN, ...

Oversampling : SMOTE

Algorithme SMOTE (Chawla et al., 2002).

Input: Echantillon d'apprentissage S de taille $m = p + n$,

k : nombre de voisins, R taux d'exemples positif à générer

Output: Un échantillon S'

begin

 Poser $New = R \times p$: nombre d'exemples à générer Syn ,
 électionner aléatoirement New exemples parmis p et noter *setind*
 leur indice

for $i \in setind$ **do**

 chercher les $k - pp$ positifs de l'exemple p_i ,
 électionner aléatoirement l'un des plus plus proches voisins

rNN_i **for attributs attr de** p_i **do**

 choisir un nombre $\alpha \in [0, 1]$,

 poser $newattr = \alpha p_i[attr] + (1 - \alpha)rNN_i[attr]$

 poser $Syn_i[attr] = newattr$

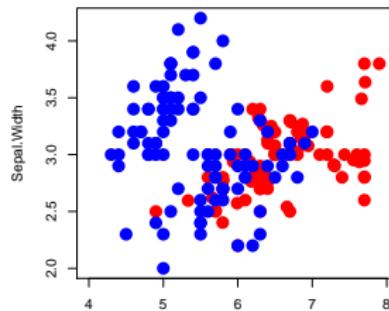
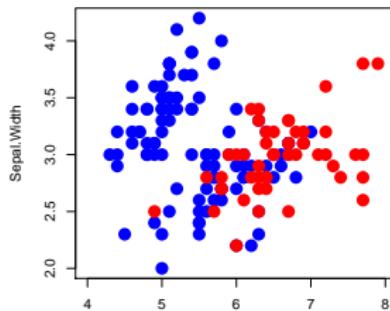
 Poser ensuite $S' = S \cup Syn_i$

 Poser ensuite $S' = S$

return S'

Oversampling : SMOTE

```
1 library(smotefamily)
2 data(iris)
3
4 data <- iris[, c(1, 2, 5)]
5 data$Species <- factor(ifelse(data$Species == "virginica","rare","common"))
6 plot(data[,c(1,2)], col = ifelse(data$Species == "rare", "red","blue"), pch=16, cex = 2,
      xlim = c(4,8), ylim = c(2,4.2) )
7
8 balanced.data <- SMOTE(data[,c(1,2)], data[,3] ,K =5, dup_size = 1)
9 new_data = balanced.data$syn_data
10 bdata = balanced.data$data
11 table(data$Species)
12 table(bdata$class)
13 plot(bdata[,c(1,2)], col = ifelse(bdata$class == "rare", "red","blue"), pch=16, cex = 2 ,
      xlim = c(4,8), ylim = c(2,4.2))
```



Undersampling : Tomek Link

Algorithme Tomek Link (Tomek, 1976)

Il s'agit d'une méthode de nettoyage de données à l'image de CNN.

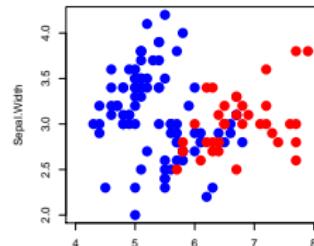
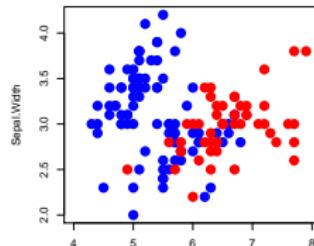
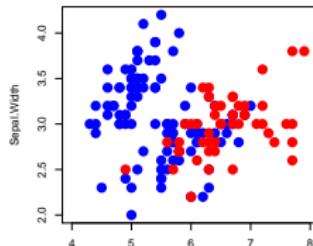
Pour se faire on considère des paires d'exemples avec des étiquettes différentes, si les deux exemples sont plus proches voisins de l'autre, on dit qu'ils forment un *lien Tomek*.

→ ces exemples se trouvent donc à une frontière et peuvent potentiellement conduire à une mauvaise classification.

On décide donc de les retirer de notre jeu de données. On peut aussi faire le choix de ne retirer que les exemples de la classe majoritaire !

Undersampling : Tomek Link

```
1 library(UBL)
2 data(iris)
3 data <- iris[, c(1, 2, 5)]
4 data$Species <- factor(ifelse(data$Species == "virginica","rare","common"))
5 plot(data[,c(1,2)], col = ifelse(data$Species == "rare", "red","blue"), pch=16, cex = 2,
       xlim = c(4,8), ylim = c(2,4.2) )
6
7 balanced.data <- TomekClassif(Species~, data, dist = "Euclidean", p = 2, Cl = c("rare",
      "common"), rem = "maj")
8 bdata_1 <- balanced.data[[1]]
9 plot(bdata_1[,c(1,2)], col = ifelse(bdata_1$Species == "rare", "red","blue"), pch=16, cex
      = 2 , xlim = c(4,8), ylim = c(2,4.2))
10
11 balanced.data <- TomekClassif(Species~, data, dist = "Euclidean", p = 2, Cl = c("rare",
      "common"), rem = "both")
12 bdata_2 <- balanced.data[[1]]
13 plot(bdata_2[,c(1,2)], col = ifelse(bdata_2$Species == "rare", "red","blue"), pch=16, cex
      = 2 , xlim = c(4,8), ylim = c(2,4.2))
```



Apprentissage dans un contexte déséquilibré

Distance et apprentissage représentations

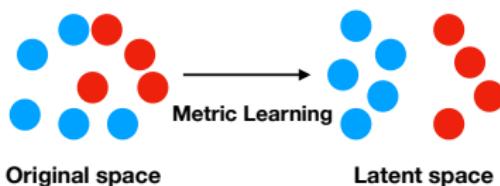
$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \mathbf{M} (\mathbf{x} - \mathbf{x}')},$$

where \mathbf{M} is PSD.

Cost-sensitive learning

$$C_{TP} = 0 \quad C_{FN} = c$$

$$C_{FP} = 1 - c \quad C_{TN} = 0$$



→ $c \simeq 1$ pour se focaliser sur la classe majoritaire.

$$\ell(y, h(\mathbf{x})) = c \cdot y \cdot (1 - h(\mathbf{x})) + (1 - c) \cdot (1 - y) \cdot h(\mathbf{x})$$

Quelques références sur le domaine : (Weinberger and Saul, 2009) (Feng et al., 2018) (Bellet and Habrard, 2015) (Bellet et al., 2013) ou encore (Davis et al., 2007).

Quelques références sur le sujet : (Elkan, 2001) (Bach et al., 2006) (Parambath et al., 2014)

Analyse (Factorielle) Discriminante Linéaire

Analyse Factorielle

Principe de l'AFD

Il s'agit d'une méthode qui consiste à trouver un espace dans lequel il est possible de mettre en évidence les différents groupes d'individus (selon leur étiquette).

Les axes discriminants sont modélisés par des hyperplans qui sont donc une combinaison linéaire des variables initiales.

Dans un contexte de classification binaire *imbalanced* mais aussi de *Big Data*, cela peut se révéler intéressant afin de créer une fonction de scoring et ainsi de réduire la taille de notre échantillon.

Principe : travailler sur les variances

Mettre les groupes d'individus en évidence

Pour cela, il faut chercher à maximiser la variance entre les classes d'individus (variance inter-classe, notée S_b) et à minimiser la variance entre les individus d'une même classe (variance intra-classe, notée S_w).

$$S_w = \frac{1}{m} \sum_{l=0}^C \sum_{i=1}^m m_l ((\mathbf{x}_i - \boldsymbol{\mu}_l)^T (\mathbf{x}_i - \boldsymbol{\mu}_l)),$$

$$S_b = \frac{1}{m} \sum_{l=0}^C m_l ((\boldsymbol{\mu}_l - \boldsymbol{\mu})^T (\boldsymbol{\mu}_l - \boldsymbol{\mu})).$$

Optimisation

L'objectif, comme pour l'ACP est de chercher des axes permettant de séparer au mieux les individus des différents groupes, en travaillant sur les variances précédemment définies.

→ **chercher le ou les axes u qui maximisent la variance inter-classes et minimise la variance intra-classe.**

Ce qui nous donne le problème d'optimisation suivant :

$$\max_{u \in \mathbb{R}^p} \frac{u^T S_b u}{u^T S_w u}.$$

Exercice

Montrer que si S_w est inversible, i.e. les variables sont indépendantes, alors ce problème peut se ramener à un problème aux valeurs propres

Optimisation : problèmes aux valeurs propres

On commence par réécrire le problème sous la forme suivante

$$\begin{aligned} \max_{u \in \mathbb{R}^p} \quad & \frac{u^T S_b u}{u^T S_w u}, \\ \text{s.t.} \quad & u^T S_w u = Id. \end{aligned}$$

On ajoute ici une contrainte sur la nature des axes (normalisation des axes).
On se rappelle maintenant de son cours d'optimisation *convexe* sous
contrainte.

Optimisation : problème aux valeurs propres

Le Lagrangien s'écrit

$$L(u, \lambda) = u^T S_b u - \lambda(u^T S_w u - Id).$$

Supposons que u et λ soient solutions du problème d'optimisation, en particulier nous $\frac{\partial L}{\partial u} = 0$, soit

$$S_b u - \lambda S_w u = 0.$$

Ce qui, étant donnée que S_w est inversible, nous ramène à la résolution du problème

$$S_w^{-1} S_b u = \lambda u.$$

Principe

On va donc rechercher les valeurs propres et vecteurs propres de la matrice $S_w^{-1}S_b$.

Le vecteur propre u avec la plus grande valeur propre λ_{\max} sera appelée **premier axe discriminant**. C'est l'axe qui permettra de séparer un premier groupe d'individus de tous les autres.

Remarque

De façon générale, le nombre d'axes discriminants est égal au **nombre de classes $C - 1$** , plus précisément, on peut montrer qu'il est inférieur à $\inf(d, C - 1)$, où d est la dimension de l'espace de nos données.

En pratique

Dans un contexte de classification avec un nombre conséquent de données, il est intéressant d'utiliser cette approche pour réduire la taille de son échantillon en écartant un nombre important de données "non intéressantes".

On utilisera la fonction *lda* de la librairie MASS.

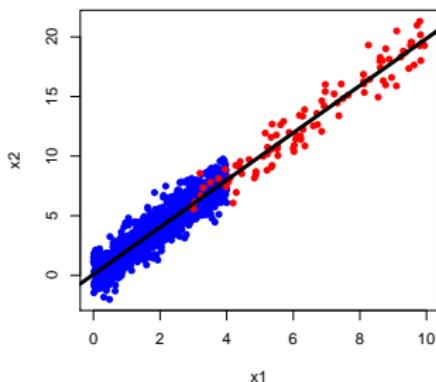
```
1 library(MASS)
2 model <- lda( Y~Var1 + Var2 + Var3 = ... , data = mydata)
3
4 model$scaling      # permet de recuperer les coefficients pour l'Analyse discriminante et
   donc le calcul du score
```

Le modèle nous donne ensuite les "poids" de chaque variable pour chaque axe permettant ainsi d'attribuer un score à nos données.

En pratique

Voilà ce que cela donnerait sur un petit exemple avec deux classes en deux dimensions

```
1 x1 <- c(runif(1000,0,4),runif(100,3,5))
2 x2 <- 2*x1 + rnorm(1100,0,1)
3 y = c(rep(0,1000),rep(1,100))
4 mydata <- data.frame(x1,x2,y)
5 model <- lda(y~x1+x2)
6
7 plot(x1,x2,col=c("blue","red")[y+1], pch = 16)
8 abline(a=o, b=p, lwd=4)
```



Boosting

Principe

Idée du boosting

Combiner plusieurs modèles qui ont un **faible** pouvoir prédictif, *i.e.* qui font à peine mieux que l' aléatoire afin de créer un modèle plus **fort** et ce, quelque soit la distribution de nos données.

On souhaite construire des modèles plus **forts, robustes**, *i.e.* des modèles, qui, s'ils sont faiblement perturbés, ne conduisent pas à une modification importante des résultats.

Cette idée du boosting répond à deux grands principes de la théorie de l'apprentissage PAC introduit à la fin des années 80' - **strong et weak PAC learnability** (Kearns and Valiant, 1994).

PAC Learnability

Strong PAC learnability

Une classe C est *strongly PAC learnable* avec une classe d'hypothèses \mathcal{H} s'il existe un algorithme \mathcal{A} tel que pour tout $c \in C$, pour toute distribution D , pour tout $\varepsilon \in (0, 1/2)$ et $\delta \in (0, 1/2)$ et ayant accès à un nombre d'exemples polynomial (en ε^{-1} et δ^{-1}) tirés de façon *i.i.d.* de D et étiquetés par c ; \mathcal{A} apprend une hypothèse $h \in \mathcal{H}$ telle que $\text{err}(h) \leq \varepsilon$ avec probabilité au moins $1 - \delta$.

Weak PAC learnability

Une classe C est *weakly PAC learnable* avec une classe d'hypothèses \mathcal{H} s'il existe un algorithme \mathcal{A} et une valeur $\gamma > 0$ tels que pour tout $c \in C$, pour toute distribution D , pour tout $\delta \in (0, 1/2)$ et ayant accès à un nombre d'exemples polynomial (en ε^{-1} et δ^{-1}) tirés de façon *i.i.d.* de D et étiquetés par c ; \mathcal{A} apprend une hypothèse $h \in \mathcal{H}$ telle que $\text{err}(h) \leq 1/2 - \lambda$ avec probabilité au moins $1 - \delta$.

PAC Learnability

On voit clairement qu'être **strongly PAC learnable** implique d'être **weakly PAC learnable** en regardant bien les énoncés et les résultats attendus sur les erreurs du classifieur.

Toute la question est maintenant de savoir s'il existe des algorithmes qui permettent à partir d'apprenants faibles issus d'un ensemble d'hypothèses \mathcal{H} de construire un nouvel espace d'hypothèse \mathcal{H}' dans lequel nous pourrions avoir un apprenant (ou une hypothèse) fort(e).

Algorithme

Un algorithme de boosting est donc capable, à partir d'apprenants **faibles** de générer un apprenant **fort**. Un exemple d'un tel algorithme de boosting est **ADABOOST**.

Input: Echantillon d'apprentissage S de taille m ,
un nombre T de modèles
Output: Un modèle $H = \sum_{t=0}^T \alpha^{(t)} h^{(t)}$

begin

Distribution uniforme $D_i^{(0)} = \frac{1}{m}$

for $t = 1, \dots, T$ **do**

Apprendre un classifieur $h^{(t)}$ à partir d'un algorithme \mathcal{A}

Calculer l'erreur $\varepsilon^{(t)}$ de l'algorithme.

if $\varepsilon^{(t)} > 1/2$ **then**

| Stop

else

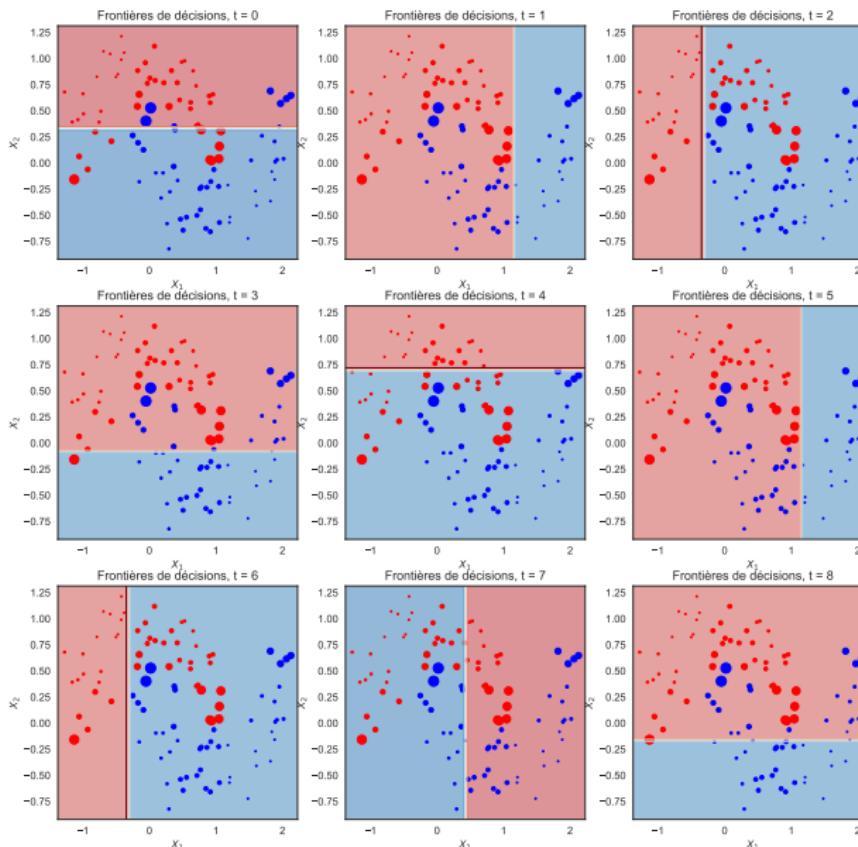
| Calculer $\alpha^{(t)} = \frac{1}{2} \ln \left(\frac{1 - \varepsilon^{(t)}}{\varepsilon^{(t)}} \right)$

| $D_i^{(t)} = D_i^{(t-1)} \frac{\exp(-\alpha^{(t)} y_i h^{(t)}(\mathbf{x}_i))}{Z^{(t)}}$

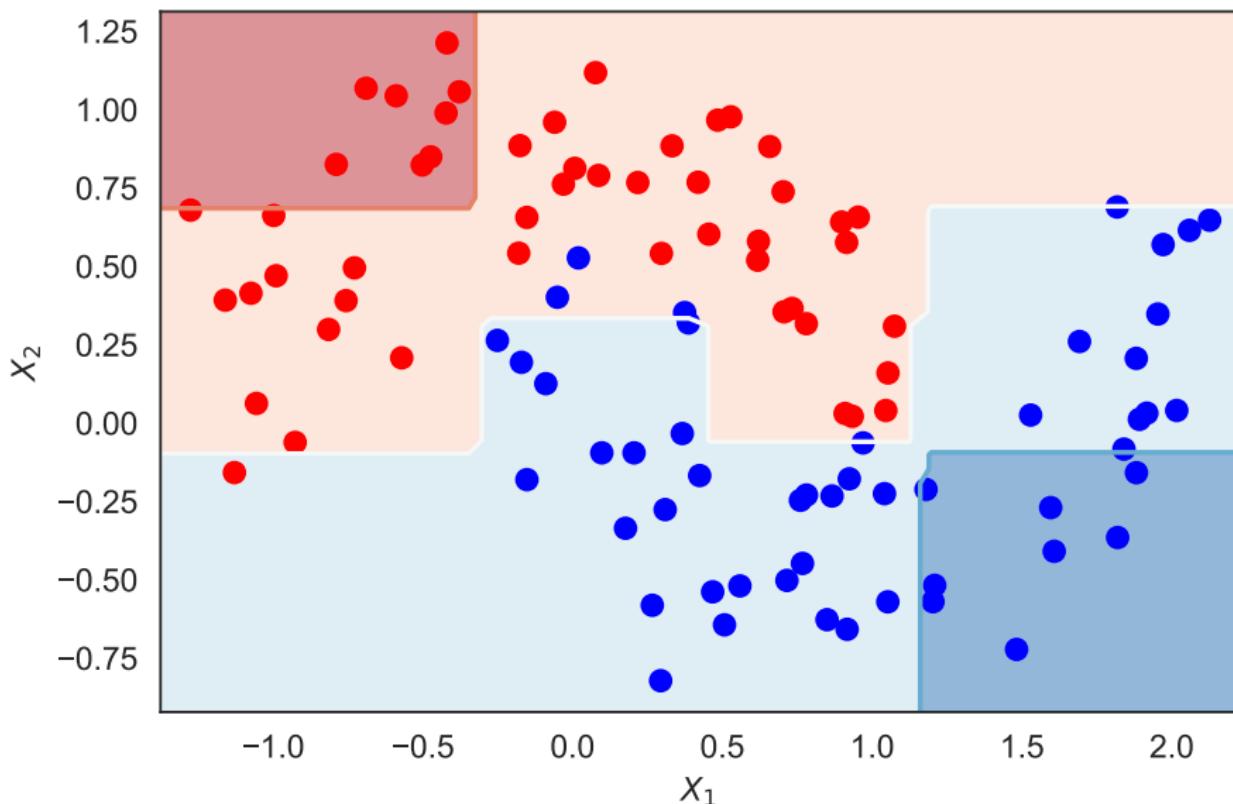
Poser $H^{(t)} = \sum_{t=0}^T \alpha^{(t)} h^{(t)}$

return $H^{(t)}$

Algorithme



Algorithme



Algorithme

- Prédiction : $\text{sign}(H^{(t)}(\mathbf{x}_i)) = \text{sign} \left(\sum_{t=0}^T \alpha^{(t)} h^{(t)}(\mathbf{x}_i) \right)$.
- Simple d'utilisation de par sa construction
- Chaque hypothèse $h^{(t)}$ vérifie l'hypothèse *weakly PAC learnable*
- Il fournit des garanties sur la convergence de l'erreur d'entraînement (a fortiori, sur l'erreur en généralisation ou en phase de test) !

Théoreme

Notons $\gamma_t = \frac{1}{2} - \varepsilon^{(t)} = \frac{1}{2} - \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} [H^{(t)}(\mathbf{x}) \neq c(\mathbf{x})]$, i.e. γ_t quantifie combien nos hypothèses faibles $h^{(t)}$ font mieux que le classifieur aléatoire. Soit $H^{(t)}$ une hypothèse obtenu par Adaboost après T itérations et γ_t .

Alors

$$\text{err}(h) \leq \exp \left(-2 \sum_{t=1}^T \gamma_t \right).$$

Algorithme

Plan de la preuve (en exercice)

1. Exprimer $D_i^{(T+1)}$ en fonction de $D_i^{(0)} = \frac{1}{m}$.
2. Montre que $err(h) \leq \prod_{t=0}^T Z_t$ en utilisant ce qui précède
3. Calculer la valeur de $\alpha^{(t)}$ optimale à chaque itération
4. Montrer que $Z_t = 2\sqrt{\varepsilon^{(t)}(1 - \varepsilon^{(t)})}$
5. Conclure

A partir de ce résultat, il est possible d'obtenir des bornes en généralisation sur l'erreur de l'algorithme Adaboost (Mohri et al., 2012)

Bilan

On montre que le boosting :

- est performant (à la fois en théorie et en pratique)
- capable de retourner des hypothèses non linéaires à partir d'hypothèses linéaires → **création de non linéarité**

Mais qu'en est-il dans un contexte de Big Data ? Est-ce toujours envisageable d'utiliser une telle procédure ?

La réponse est **oui** mais pas forcément en utilisant un algorithme comme Adaboost et pas avec n'importe quel type de classifieur ... si on retournait sur les arbres ?

Gradient Tree Boosting

Pouquoi utiliser des arbres ?

- Nous avons vu plus tôt que les arbres sont capables d'apprendre de façon extrêmement précises les données (biais très faible)
- De plus, les combiner, en utilisant des méthodes d'échantillonnage, permettait de fortement réduire la variance des résultats (gain en stabilité)
- C'est un algorithme qui est peu gourmand en temps de calcul
- Mais qui est aussi facilement optimisable car parallélisable ! On ne fait que manipuler des listes.

Pour toutes ces raisons, c'est un algorithme que l'on pourrait largement privilégier dans ce contexte ... mais si on pouvait faire encore mieux ?

Pouquoi ne pas faire du boosting avec des arbres ?

- Après tout, on a montré que combiner plusieurs modèles ensembles permettait de créer un modèle plus fort
- Le processus itératif est relativement simple à mettre en œuvre et se concentre sur les erreurs effectuées par le modèle précédent.

Quid de cette histoire d'*hypothèses faibles* évoquée dans la présentation du boosting ?

On va simplement prendre des arbres moins profonds ! Donc avec un biais plus grand mais qui vont s'apprendre plus vite ! On va faire du *Gradient (Tree) Boosting*

Gradient Boosting

Algorithme de Gradient Boosting tel que présenté par Friedman (2000)

Input: Ensemble $S = \{\mathbf{x}_i, y_i\}_{i=1}^m$, une loss ℓ , nombre d'itérations T

Output: Un modèle $\text{sign}\left(H^0(\mathbf{x}) + \sum_{t=0}^T \alpha^t h_{a^t}(\mathbf{x})\right)$

begin

Hypothèse initiale $H^0(\mathbf{x}_i) = \arg \min_{\rho \in \mathbb{R}} \sum_{i=1}^m \ell(y_i, \rho) \quad \forall i = 1, \dots, m$

for $t = 1, \dots, T$ do

 Calculer les pseudo-résidus :

$$\tilde{y}_i = -\frac{\partial \ell(y_i, H^{t-1}(\mathbf{x}_i))}{\partial H^{t-1}(\mathbf{x}_i)}, \quad \forall i = 1, \dots, m$$

 Apprendre un modèle pour les fitter :

$$a^t = \arg \min_{a \in \mathbb{R}^d} \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$$

 Apprendre le poids du classifieur :

$$\alpha^t = \arg \min_{a \in \mathbb{R}^+} \sum_{i=1}^m \ell(y_i, H^{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$$

 Mise à jour $H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$

return H^t

Quelle différence avec le boosting classique ?

Gradient Boosting

En fait, il n'y en a pas ... enfin ce n'est pas tout à fait vrai ! Le gradient boosting peut être vu comme une généralisation plus flexible de notre algorithme *Adaboost* précédemment évoqué.

- *Adaboost* est basé sur la *logistic loss* alors que n'importe quelle loss est utilisable pour la *gradient boosting*.
- *Adaboost* travail directement sur le poids des données pour corriger les erreurs alors que le *gradient boosting* utilise le gradient

Le gradient boosting se présente en fait comme un algorithme d'optimisation dans un espace fonctionnel, l'espace des *prédictions*.

Il fonctionne comme un algorithme de descente de gradient à pas optimal (ou le plus profond).

Gradient Boosting

Algorithme de Gradient Boosting d'un point de vue optimisation

Input: Ensemble $S = \{\mathbf{x}_i, y_i\}_{i=1}^m$, une loss ℓ , nombre d'itérations T

Output: Un modèle $\text{sign}\left(H^0(\mathbf{x}) + \sum_{t=0}^T \alpha^t h_{a^t}(\mathbf{x})\right)$

begin

Initialisation : $H^0(\mathbf{x}_i) = \arg \min_{\rho \in \mathbb{R}} \sum_{i=1}^m \ell(y_i, \rho) \quad \forall i = 1, \dots, m$

for $t = 1, \dots, T$ **do**

Calcul du gradient : $\tilde{y}_i = -\frac{\partial \ell(y_i, H^{t-1}(\mathbf{x}_i))}{\partial H^{t-1}(\mathbf{x}_i)}, \quad \forall i = 1, \dots, m$

Apprentissage d'un modèle qui approxime le gradient :

$a^t = \arg \min_{a \in \mathbb{R}^d} \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$

Pas d'apprentissage optimal :

$\alpha^t = \arg \min_{\alpha \in \mathbb{R}^+} \sum_{i=1}^m \ell(y_i, H^{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$

Mise à jour du modèle : $H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$

return H^t

Regardons maintenant un exemple avec l'algorithme XGBoost (Chen and Guestrin, 2016).

XGBoost

A partir de maintenant on considère le problème d'optimisation suivant :

$$\min \sum_{i=1}^m L(y_i, \hat{y}_i) + \beta L + \frac{\lambda}{2} \sum_{j=1}^L (f_j(t))^2, \quad (1)$$

où βL et $\frac{\lambda}{2} \sum_{j=1}^L (f_j(t))^2$ sont deux termes de régularisation contrôlant le nombre de feuilles et le poids de chaque feuille de l'arbre.

Le problème (1) est équivalent au problème suivant, en utilisant le principe de l'additivité des modèles

$$\min \sum_{i=1}^m L(y_i, \hat{y}_i^{(t-1)} + h^{(t)}(\mathbf{x}_i)) + \beta L + \frac{\lambda}{2} \sum_{j=1}^L (f_j^{(t)})^2. \quad (2)$$

XGBoost

L'idée consiste ensuite à utiliser simplement une approximation d'ordre 2 de la fonction à optimiser :

$$\begin{aligned} L(y_i, \hat{y}_i^{(t-1)} + h^{(t)}(\mathbf{x}_i)) &\simeq L(y_i, \hat{y}_i^{(t-1)}) \\ &+ h^{(t)}(\mathbf{x}_i) \nabla h^{(t)}(\mathbf{x}_i) + \frac{1}{2} h^{(t)}(\mathbf{x}_i)^2 \nabla^2 h^{(t)}(\mathbf{x}_i) \end{aligned}$$

La fonction $h^{(t)} = \left(h_j^{(t)}\right)_{j=1,\dots,L}$ que l'on souhaite apprendre est un ensemble de feuilles, on veut donc déterminer la valeur optimale à donner à chaque feuille

$$\min \sum_{i \in I_j} \left[h^{(t)}(\mathbf{x}_i) \nabla h^{(t)}(\mathbf{x}_i) + \frac{1}{2} (\lambda + \nabla^2 h^{(t)}(\mathbf{x}_i)) h^{(t)}(\mathbf{x}_i)^2 \right] \quad (3)$$

XGBoost

Si on cherche à résoudre l'équation (3) en résolvant l'*équation d'Euler*, i.e. en utilisant la convexité de la fonction et en cherchant pour quelle valeur de $f_j^{(t)}$ la fonction s'annule, on trouve

$$f_j^{(t)} = -\frac{\sum_{i \in I_j} \nabla h(\mathbf{x}_i)}{\sum_{i \in I_j} \nabla^2 h(\mathbf{x}_i) + \lambda}$$

Exercice

- 1) En déduire la valeur optimale du problème (3).
- 2) Déterminer le critère de séparation d'un nœud, à l'image de ce que nous avons vu avec les arbres de décisions.

En pratique

L'utilisation de **XGBoost** sous R nécessite l'utilisation d'un certain format pour les données. Il n'est pas possible d'utiliser des tableaux ou matrices classiques et convient de faire la transformation suivante.

```
1 library(xgboost)
2 xgb.data.train <- xgb.DMatrix(Data[, "features"], label = Data[, "label"])
```

Il ne reste ensuite qu'à voir comment apprendre le modèle.

En pratique

Présentation des différents paramètres d'apprentissage de la fonction.

```
1 bst <- xgb.train(data = xgb.data.train
2 , params = list(objective = "binary:logistic"
3 , eta = 0.07
4 , max.depth = 8
5 , gamma = 0.1
6 , min_child_weight = 20
7 , subsample = 0.7
8 , colsample_bytree = 0.7
9 , nthread = 7
10 , eval_metric = "auc"
11 , lambda = 2
12 , alpha = 2
13 )
14 , watchlist = list(eval = xgb.data.valid, train=xgb.data.train)
15 , nrounds = 300
16 , early_stopping_rounds = 40
17 , print_every_n = 20
18 )
19
# Pour ensuite tester le modèle en prenant les meilleures performances en validation
20
21 predict( bst,
22     newdata = test_data,
23     ntree limit = bst$bestInd)
```

Algorithme

C'est un algorithme simple d'utilisation et très facile à implémenter et surtout très flexible. En effet, les fonctions **objective** et **eval-metric** sont totalement personnalisables, vous pouvez mettre ce que vous voulez.

vous donne des fonctions objectives classiques comme la *logistic loss* pour des problèmes de classification binaire ou multi-classe, ranking, ... De même les performances de votre modèle peuvent se faire selon plusieurs critères : AUC ROC - AUC PR - Binary error - Multi-class error

Algorithme

Un exemple en attribuant un coût à chaque exemple (cost-sensitive learning)

Personnalisation "objective"

```
1 myobjective <- function(pred,dtrain){  
2  
3   label <- getinfo(dtrain, "label")  
4   amount <- getinfo(dtrain, "weight")  
5  
6   cost_FP <- rho*r*amount - p  
7   cost_FN <- (cost(amount) - r) * amount  
8   cost_TP <- rep(0, length(label))  
9   cost_TN <- r*amount  
10  
11  s <- function (costs)  
12  
13  grad <- function(costs, s)  
14  hess <- function (costs, s)  
15  
16  return(list(grad=grad, hess = hess))  
17}
```

Il est important, pour la fonction objective que vous définissez, de retourner les dérivés premières et secondes de votre fonction objective.

Algorithme

Un exemple en attribuant un coût à chaque exemple (cost-sensitive learning)

Personnalisation "eval-metric"

```
1 mymetric <- function(pred,dtrain){  
2  
3   label <- getinfo(dtrain, "label")  
4   amount <- getinfo(dtrain, "weight")  
5  
6   cost_FP <- rho*r*amount - p  
7   cost_FN <- (cost(amount) - r) * amount  
8   cost_TP <- rep(0, length(label))  
9   cost_TN <- r*amount  
10  
11  s <- function (costs)  
12  
13  marge <- function (costs, pred)  
14  
15  return(list( metric = "marge", value = marge))  
16}
```

Il faudra ici renvoyer une liste contenant le nom de la métrique ainsi que sa valeur.

Pour finir

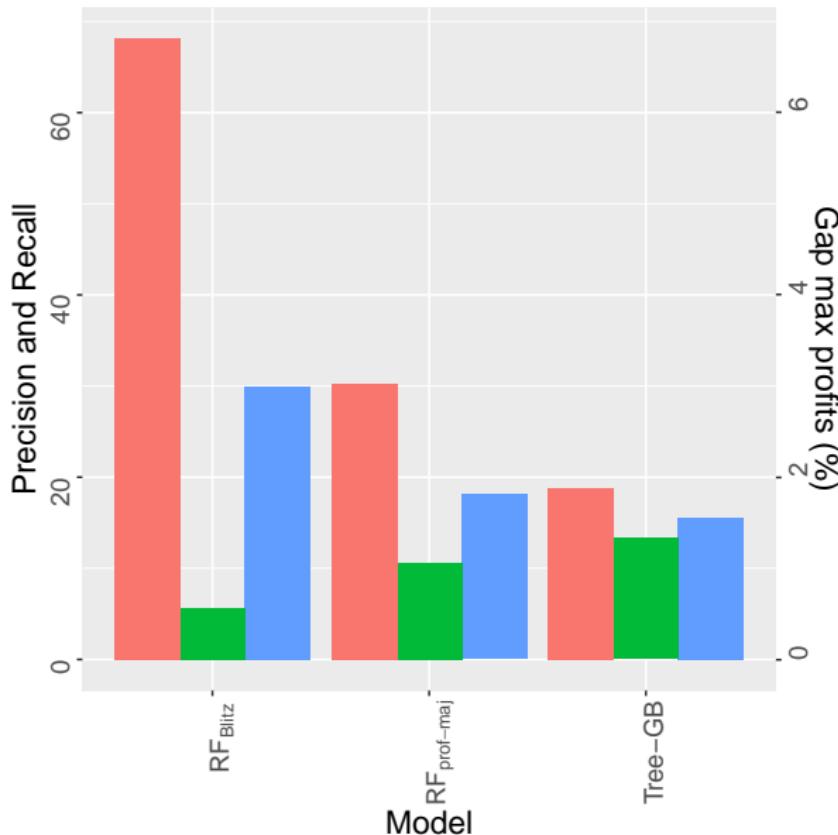
Modèle de marge

Pour information, l'utilisation des méthodes cost-sensitive ici est employée afin de maximiser la marge d'une enseigne en prenant en compte :

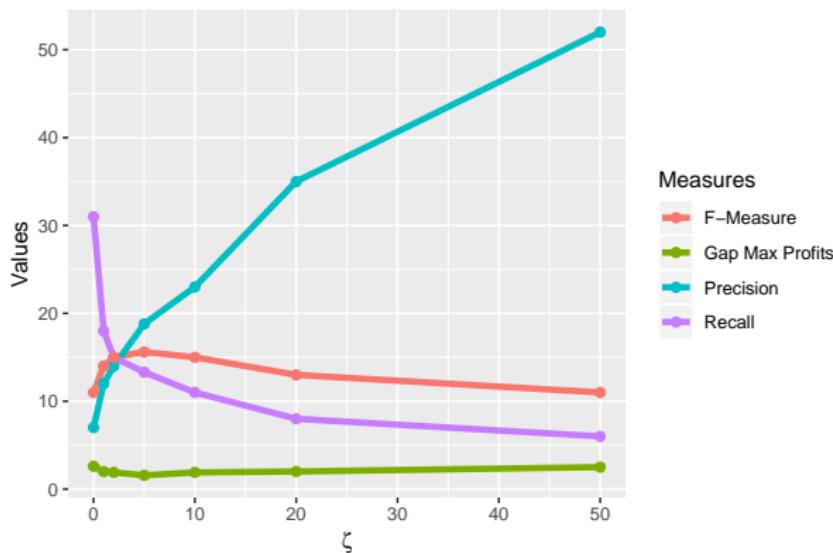
- un modèle de marge
- l'étiquette des transactions (fraude / non-fraude)
- le montant de la transaction

On a souhaité comparer ce que peuvent donner les forêts aléatoires (en modifiant le critère de split) et le gradient tree boosting dans l'optimisation de cette marge.

Pour finir



Pour finir



References |

- Bach, F. R., Heckerman, D., and Horvitz, E. (2006). Considering cost asymmetry in learning classifiers. *Journal of Machine Learning Research*, 7(Aug):1713–1741.
- Bellet, A. and Habrard, A. (2015). Robustness and generalization for metric learning. *Neurocomputing*, 151:259–267.
- Bellet, A., Habrard, A., and Sebban, M. (2013). A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357.
- Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM.
- Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. S. (2007). Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM.
- Elkan, C. (2001). The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, pages 973–978, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

References II

- Feng, L., Wang, H., Jin, B., Li, H., Xue, M., and Wang, L. (2018). Learning a distance metric by balancing kl-divergence for imbalanced datasets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.
- Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.
- Kearns, M. and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95.
- Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.
- Parambath, S. P., Usunier, N., and Grandvalet, Y. (2014). Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems (NIPS-14)*, pages 2123–2131.
- Tomek, I. (1976). Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, 6:769–772.
- Weinberger, K. Q. and Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244.