1ère année 2018

# Transfert d'hypothèse par projection entre espaces

Un sujet intéressant de Pierre-Alexandre Murena

Florian Bonnard Christophe Vuong

Machine Learning

# Table des matières

Chapitre 1

# Introduction

**Vocabulaire et notations adoptés**

Un jeu de données de cardinal $N$ sera représenté par des vecteurs $(X_i)_{i=1...N}$ dont les composantes sont les features... Pour des images de chiens et de chats, cela peut être la couleur des pixels donnée par une valeur pour chaque pixel (en niveau de gris, sinon x3 en RVB). On pourrait trouver une représentation vectorielle dans $\mathbb{R}^{nombre\ de\ pixels}$.
On pourrait aussi l'exprimer comme une matrice, mais on part de cette description mathématique des données.

On parlera de domaines pour désigner l'espace vectoriel de représentation d'un type de données (exemple : le domaine des images en 8x8 pixels des chiffres décimaux).

## 1.1 Positionnement dans le machine learning

### 1.1.1 Apprentissage supervisé :

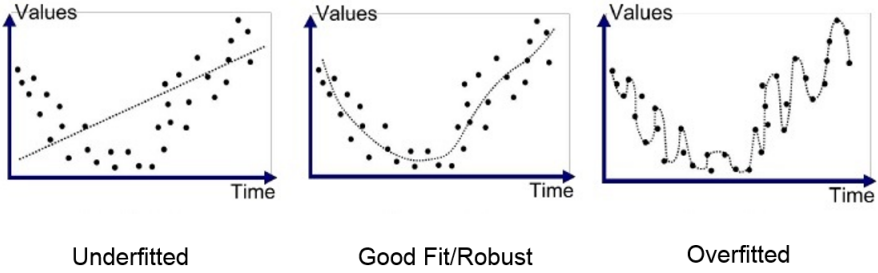
L'estimateur est appris dans le cadre d'exemples annotés (avec des labels). C'est-à-dire qu'on a accès aux classes des données pour le constituer.

**Classificateur binaire :** Cas particulier d'estimateur où l'ensemble d'arrivée est de cardinal 2, par exemple $\{-1, 1\}$. Dans les bases de données que l'on va étudier, on prendra par commodité des classificateurs dans

les espaces, définis ainsi :

$$h : \mathbb{R}^d \longmapsto \{-1, 1\} \tag{1.1}$$

**2 cas particuliers dans l'apprentissage :**




Soit $(X_i, Y_i)_{i=1,\dots,N}$ un jeu de données avec label. $Y_i \in \{-1, 1\}$ indique la classe à laquelle appartient la donnée, supposée connue dans le cas d'apprentissage supervisé, ce qui n'est pas le cas en non supervisé (exemple : clustering).

Pour qu'un classificateur soit considéré robuste dans un jeu de données, il faut qu'il sépare bien les données, même quand on doit ajouter de nouvelles données de même nature dans le set.

L'overfitting ou surapprentissage correspond au cas où l'apprentissage donne un classificateur qui colle trop aux données à disposition au point de voir ces performances se détériorer si on élargit la base de données comme l'illustre le schéma ci-dessus.

### 1.1.2 Méthode générale pour vérifier les performances de cette technique : cross-validation

Etant donné une base de données, la démarche classique en machine learning est de scinder notre base de données afin de former un **training set** de données et d'autres sets appelés **validation sets** tous labellisés pour tester la robustesse de notre classificateur à l'overfitting (surapprentissage). En effet, on suppose que le classificateur s'entraîne que sur le training set.

Puis on l'applique pour des données d'un validation set à priori indépendantes du training set. Si on cherche à avoir un classificateur qui va avoir de bonnes performances en dehors de notre training set on va essayer de limiter l'erreur en cross-validation (tout en évitant d'entraîner notre classificateur suivant ce set de validation ce qui entrainerait de nouveau la même problématique que pour un autre set).

L'idéal sera alors d'avoir une erreur de cross-validation de l'ordre de grandeur de celle sur le training set.

## 1.2 Cadre global de nos travaux

L'apprentissage par transfert est différent de l'apprentissage classique dans le sens où dans ce cadre il serait possible de ne pas apprendre depuis le début avec un jeu de données. Supposons que l'on ait une base de données $(X_i^s)_{i=1,\dots,N_s}$ sur laquelle a été déjà entraîné un classificateur qui a de bonnes performances. Il peut être intéressant de se servir de cette bonne classification pour faire de la classification sur un second domaine, moyennant une relation liant ces 2 domaines. $(X_i^{tar})_{i=1,\dots,N_s}$.

Pour quelles raisons ?

— Manque de données pour entraîner un classificateur pour discriminer les $(X_i^{tar})_{i=1,N_s}$

— Entraînement d'un classificateur sans succès escompté

Par exemple, si on considère des images de chiens/chats pour le premier domaine, et de hommes/femmes pour le deuxième. D'un point de vue visuel, il y a une corrélation par exemple entre les rapports de taille des 2 classes des 2 domaines visuellement parlant.

Nos travaux ne prétendent pas s'intéresser à maximiser l'exploitation de la corrélation entre les 2 domaines. En revanche, on peut montrer qu'avec peu d'information sur le deuxième domaine, et son lien avec le premier, on peut, moyennant certaines hypothèses évaluer la classification à l'aide du jeu de données $X_s$ bien connu trouver mieux que d'"inventer" une corrélation, notamment avec une hypothèse prise aléatoirement comme on peut le voir dans la suite.

### 1.2.1 Cadre de travail du transfert

— Domaine source : $(X_i^S, Y_i^S)_{i=1,\dots,N_S}$ où $X_i^S \in \mathbb{R}^n$

— Domaine cible : $(X_i^{tar}, Y_i^{tar}{}_{i=1,\dots,N_t}$ où $X_i^{tar} \in \mathbb{R}^p$

On notera $D_s$ et $D_t$ respectivement les ensembles de données des 2 domaines.

On considère deux classes (classification binaire) :

— 2 classes $\{-1, 1\}$

— $Y_i \in \{-1, 1\}$ (-1, et 1 désignent alors les labels des classes)

— On dispose de bon(s) classificateur(s) $H_S$ pour le jeu de données source que l'on peut étendre à l'espace dans lequel vit les vecteur data. On note $\mathcal{H}_S$ l'ensemble des classificateurs sources.

On a **peu de données** dans le domaine cible et on veut utiliser la connaissance du domaine source pour avoir un classificateur meilleur que

dans le cas d'apprentissage classique.

L'idée est d'utiliser des projections pour se ramener du domaine cible au domaine source, et de générer des classifieurs en fonction de ces projections $\in \mathcal{P}$ et des classifieurs dans $\mathcal{H}_S$.

**On obtient alors un ensemble de classifieurs pour le domaine cible :**

$$\aleph_t = \{h_s \circ \Pi | \in \mathcal{P}; h_S \in \mathcal{H}_S\} \tag{1.2}$$

On prend effectivement les meilleurs classificateurs du domaine source, on ne s'occupe pas de comment les déterminer dans le cadre strict de notre projet. En pratique, il faut néanmoins entraîner un classificateur sur le domaine source soi-même, ce qui peut se faire comme décrit en annexe[A].

**Problématique** Que sait-on sur cet ensemble de classificateurs et en quoi cela va nous aider à trouver un bon classificateur en target, sans faire le processus complet d'apprentissage sur $D_t$ ?

Chapitre 2

# Le boosting, la clé du transfert ?

### 2.0.1 Principe du transfert sans boosting

Une idée naïve serait de sélectionner les meilleurs éléments de $\aleph_T$ obtenus minimisant le risque :

$$R(h) = R(H_S \circ \Pi) = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathbb{I}(H_s \circ \Pi(X_i^{tar}) \neq Y_i^{tar}) \tag{2.1}$$

Mais comment s'assurer d'avoir les meilleurs classificateurs sachant qu'il y a certainement une multitude de projections qui peuvent apporter un risque faible, de ce fait il va falloir générer un nombre incalculable de projections qui peuvent être non linéaires, ce qui semble un véritable casse-tête, car pour appliquer peut-être par la suite des techniques d'optimisation pour éventuellement trouver des bonnes projections au sens du risque des classificateurs cibles ainsi formés, il faudrait une première idée de la forme de la projection.

### 2.0.2 Modèle linéaire pour le transfert

Forme de la projection $\Pi$ :

$$X^s = \Pi(X^{tar}) = AX^{tar} + B \tag{2.2}$$

où $A \in \mathcal{M}_{n,p}(\mathbb{R})$ et $B \in \mathcal{M}_n(\mathbb{R})$

Cette forme semble la plus simple, mais pourtant, elle donne des résultats assez spectaculaires comme on peut le voir par la suite.

En pratique, il faudra donc faire varier A et B. On pourrait optimiser le risque (2.1) en fonction de ces paramètres, mais on va adopter une méthode qui n'exclut pas forcément cette première idée de méthode.

### 2.0.3 Principe du boosting et application dans le transfert d'hypothèse

Dans la plupart des articles sur le transfert d'hypothèse[1][3], on ne parle pas de boosting qui est pourtant assez incontournable en machine learning.

— Prendre plein de classifieurs faibles (un peu meilleur que de décider au hasard)
— Générer un "meilleur" classificateur à partir de ces derniers via AdaBoost (Gödel,2003)

Le pseudo-code en détail se trouve en annexe[A].


**AdaBoost à la rescousse** A partir de notre ensemble de classificateurs dans $\aleph_t$ (1.2), on va vouloir déterminer au moins un bon classificateur du domaine cible.

**Petite astuce :** On peut s'assurer que les classificateurs dans cet ensemble soient tous de risque (2.1) $R(h_{tar} \leq 0.5)$. En effet, il suffirait de le calculer et de prendre l'opposé du classificateur si le risque est supérieur à 0.5, ou de le laisser tel quel sinon.

**Une première idée avec boosting :** Les propriétés d'AdaBoost permettent a priori de combiner tous les classificateurs de $\aleph_t$ obtenus avec cette astuce, et ce via adaBoost. Si on se restreint au modèle linéaire pour les projections. Il semblerait qu'il suffise de déterminer tous les classificateurs par projections possibles notamment au hasard et de les booster. En pratique, on se limitera à un nombre prédéfinies de matrices A et B générées aléatoirement.

Néanmoins, cette idée semble conceptuellement difficile d'accès. Il y a bien sûr des cas de faibles classificateurs pour lesquels en entrée d'AdaBoost on ait les meilleures performances comme décrit dans [A]. Or, une première limite d'AdaBoost est **sa robustesse à l'overfitting**, car plus on améliore des performances de classificateurs, plus il y a risque d'overfitting. On ne peut pas évaluer théoriquement cette limite qui dépend fortement des cas d'utilisation. Notre démarche motivée par la structure des jeux de données nous mène à utiliser une hypothèse de taille qu'est la linéaire séparablité pour établir notre solution qui fonctionne globalement mieux dans tous les jeux de données envisagés.

Chapitre 3

# Enrichissement de la démarche

## 3.1 Une propriétés courante sur les datasets : la linéaire séparabilité

Notre premier jeu de domaines sera **Iris et WBDC**. Il se trouve qu'ils sont linéairement séparables, on verra que la corrélation réside dans ce caractère linéairement séparable. En effet, à première vue les 2 domaines n'ont aucun rapport que ce soit à l'oeil ou en regardant les valeurs des données.

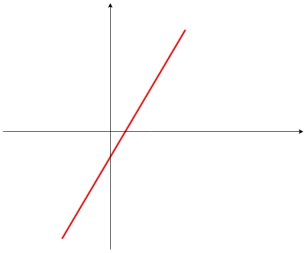
On qualifie un domaine ($\mathbb{R}^n$) à 2 classes de linéairement séparable si ces 2 classes vérifient la relation :

Linéaire séparabilité :

Pour la classe de label -1 : $\sum_{i=1}^{n} \alpha_i x_i < k$

Pour la classe de label 1 : $\sum_{i=1}^{n} \alpha_i x_i > k$

où $k \in \mathbb{R}$ et $(\alpha_i)_i \in \mathbb{R}^n$ et $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$



Ce qui précède signifie qu'il existe un hyperplan de $\mathbb{R}^n$ qui sépare le domaine en 2 classes, cet hyperplan est défini par :

$$\sum_{i=1}^{n} \alpha_i x_i = k \tag{3.1}$$

En pratique on aura affaire à des cas où la marge sur la limite est plus grande et on cherchera alors un hyperplan tel que :

**En pratique :** L'hyperplan est d'équation : $\sum_{i=1}^{n} \alpha_i x_i = k$

Il existe $C \in \mathbb{R}^+$ tel que :

Pour la classe de label -1 : $\sum_{i=1}^{n} \alpha_i x_i < k - C$

Pour la classe de label 1 : $\sum_{i=1}^{n} \alpha_i x_i > k + C$

où les hyerplans d'équations $\sum_{i=1}^{n} \alpha_i x_i = k - C$ et $\sum_{i=1}^{n} \alpha_i x_i = k + C$ contiendront au moins un point de la classe -1 et de la classe 1 respectivement.

### 3.1.1 Préprocessing

**Repartition des données dans les 2 classes** Pour éviter un phénomène de biais (le classificateur ne converge pas correctement, par exemple l'hyperplan est décallé dans le cas linéairement séparable) dans le classificateur on a :

**Eviter le biais avec l'utilisation d'arbre de décisions :** Les arbres de décisions[A] ,utilisés ici pour AdaBoost pour aider à la classification en source et en target, ont tendance à produire des biais lorsque certaines données sont trop amassées autour de certains points.

On a donc ajusté les jeux de données pour en avoir autant de données de classe -1 que de classe 1 pour les entraînements.

## 3.2 Etude de la projection dans le cadre linéairement séparable

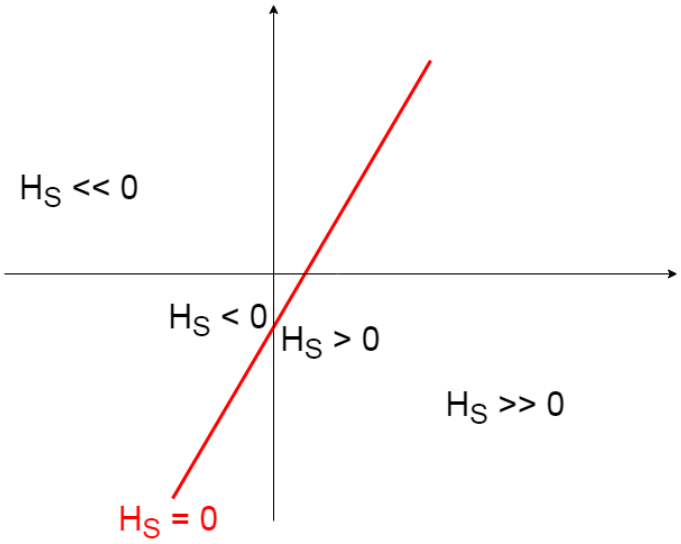
Choisir A et B aléatoirement avec chaque composante dans $\mathbb{R}$ semble clairement sous optimal, comment borner ces composantes tout en n'étant pas restreint dans nos applications ?

### 3.2.1 Avec le domaine source et cible linéairement séparables :

2 hyperplans affines $\mathbb{H}_S$ et $\mathbb{H}_T$ séparent alors respectivement le domaine source et le domaine cible :

Soient $X_{S0} \in \mathbb{H}_S$ et $X_{T0} \in \mathbb{H}_t$, on souhaite alors obtenir $h_s^{lin}(X_{S0}) = H_s(AX_{T0} + B) = 0$

où $h_s^{lin}$ désigne un estimateur qui va prendre des valeurs dans $\mathbb{R}$, la classe -1 correspond à $H_s(X) < 0$ et la classe 1 à $h_s^{lin}(X) > 0$, $h_s^{lin}(X) = 0$ étant la zone de décision.



### 3.2.2 Linéaire séparabilité : simplification du modèle linéaire

Pourquoi ne pas essayer de montrer qu'on peut prendre $X_{S0} = AX_{T0} + B$ et juste trouver une matrice A qui fonctionnerait au hasard ?

**Démonstration dans le cas n = p : (voir en annexe[A] pour le cas général**

On pose $X'_S = X_S - X_{S0}$ et $X'_T = X_T - X_{T0}$ ce qui revient à centrer les 2 hyperplans affines autour de l'origine et donc à obtenir 2 hyperplans d'équations :

$\sum_{i=1}^{n} \alpha_{Si} x_{Si} = 0$ et $\sum_{i=1}^{n} \alpha_{Ti} x_{Ti} = 0$

On peut alors prendre 2 bases orthonormées dans chaque domaine dont la normale à l'hyperplan dans le domaine est un des vecteurs.

On peut alors passer d'un domaine à l'autre par une matrice A orthogonale (changement de base orthonormées)

Donc une telle matrice permet d'avoir $X'_S = AX'_T$, càd $X_S - X_{S0} = A(X_T - X_{T0})$, càd $X_S = AX_T - AX_{T0} + X_{S0}$ et on pose $B = X_{S0} - AX_{T0}$

Avec notre équation sur B on a alors plus qu'à déterminer A :

### 3.2.3 Conséquence de la démonstration précédente

On peut prendre A avec des coefficients dans [-1,1] ou dans n'importe quel intervalle du type [-a,a] où $a \in \mathbb{R}$ en effet, il suffit juste de changer la norme des vecteurs de notre base source (ou cible) pour cela. Par ailleurs, elle est orthogonale au sens où pour $n \geq p$ : $A^T A = I_n$ telle que l'on a décrite dans la démonstration. On voit que cette caractéristique des données créée une corrélation entre les 2 jeux de données en contraignant la matrice $A_{optimal}$.

Notre hypothèse est qu'en cadrant déjà bien la matrice A, on puisse avoir un meilleur résultat avec même si le boosting appliqué ne recquiert pas de connaître quoique ce soit sur la matrice de projection.

Et au pire des cas, même si on n'approche pas ce $A_{optimal}$, avec du boosting on pourra compenser les non-linéarité qui existent dans les cas pratiques.

**Obtention de $X_{S0}$ et $X_{T0}$** Pour obtenir $X_{S0}$ et $X_{T0}$, on a implémenté l'algorithme perceptron adapté à de la classification entre 2 domaines linéairement séparables :

**Principe de l'algorithme perceptron :**

- Déterminer par évaluation des $X_i$ successifs les coefficients $\alpha_i$ et k de l'hyperplan séparant les 2 classes
- Pour son bon fonctionnement on a élaboré une fonction qui permet le préprocessing des données en alternant les données de classe -1 et les données de classes 1 (afin d'éviter que l'hyperplan tende vers un des extrêmes lors de l'évaluation des derniers $X_i$ du set) visant à éliminer le biais induit

### 3.2.4 Première démarche

- Rendre les decision stumps meilleur que la classification au hasard dans le domaine source avec l'astuce du signe. Appliquer AdaBoost pour obtenir $H_s$
- Trouver les paramètres des hyperplans respectifs avec l'algorithme perceptron.
- Générer les classificateurs target en faisant varier aléatoirement la matrice A qui fait elle varier la matrice B pour chaque classificateur fort selon 3.2.2
- Booster ces classificateurs target pour obtenir $H_t$

### 3.2.5 Un premier jeu de données et première performance

Retour sur cette paire de jeux de données :

**Source :** Le jeu de données Iris réduit à 2 classes (Iris-setosa et Iris-versicolor) est de dimension 4.

Cible : Le jeu de données WDBC sur des cellules cancéreuses pour le cancer du sein (bénine ou maline) est de dimension 30.

**Résultats** On s'appuie alors sur nos résultats précédents et on synthétise une classe Python permettant de générer des matrices A comme dit précédemment jusqu'à atteindre un certain seuil d'erreur sur le risque pour le set target puis de passer cette matrice et des matrices voisines (en terme de valeurs des coefficients) dans AdaBoost.

Sur 100 essais ciblant une erreur 0.1 (seuil de 0.1)

| Erreur moyenne sur le set de départ | Ecart-type sur l'erreur du set de départ | Erreur moyenne sur le set de cross validation | Ecart-type sur l'erreur du set de validation | Nombre moyen de matrices générées | Ecart-type sur le nombre moyen de matrices générées | Temps moyen pour obtention (en s) | Ecart-type sur le temps moyen pour obtention (en s) |
|---|---|---|---|---|---|---|---|
| 0.056 | 0.017 | 0.101 | 0.034 | 312 | 213 | 3.080 | 0.151 |

## 3.3 Extension de la première application

La justesse de la relation entre A et B n'est corhoborée que par nos résultats. De ce fait, on pourrait se demander si cette méthode peut se généraliser par exemple pour des jeux de données non linéairement séparables. On voudrait également faire des tests dans des cas où on perd cette linéaire séparabilité, pour cela faudrait-il changer nos domaines ? Et bien non comme nous allons le voir.

Prenons le cas où notre domaine cible est WDBC et éliminons la dernière dimension (ce qui revient à passer dans $\mathbb{R}^{p-1}$), on a alors :

**Relation permettant de déterminer l'appartenance à une classe :**

classe -1 : $\sum_{i=1}^{p-1} \alpha_i x_i < k - C - \alpha_p x_p$

classe 1 : $\sum_{i=1}^{p-1} \alpha_i x_i > k + C - \alpha_p x_p$

On voit que ce changement de dimension influe sur la marge par rapport à l'hyperplan, et cette marge est donnée pour $\alpha_p \geq 0$ par : $C - \alpha_p max_p - (-C - \alpha_p min_p)$ , ce qui revient à une marge du type :

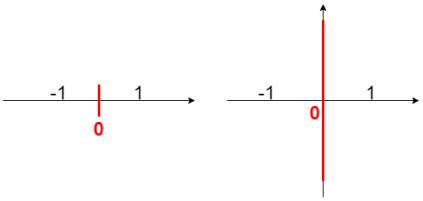
$$2C - |\alpha_p|(max_p - min_p)$$

Plus $\alpha_p(max_p - min_p)$ **est grand** plus on perd en marge en linéaire séparabilité. On peut même arriver à ne plus avoir de linéaire séparabilité si la valeur de la marge calculée s'avère être négative.

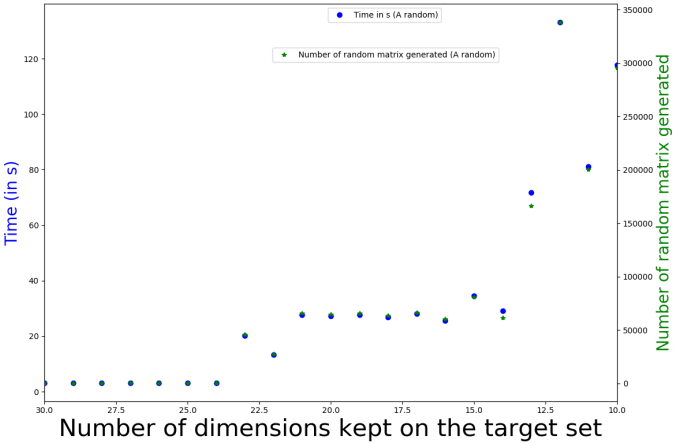
En parallèle lorsque l'on augmente la dimension du set cible tout en concervant la linéaire séparabilité, il devient plus difficile de déterminer une matrice aléatoire qui permet de bien classifier les données.

**Avec une source de dimension 1 et une cible passant de la dimension 1 à 2 :** Pour la dimension 1 : $x_S = ax_{T_0}$

Pour la dimension 2 : $x_S = ax_{T_0} + bx_{T_1}$




On a alors la dualité entre 2 phénomènes dans la réduction de dimension de l'arrivée. Un benchmark va permettre de les départager, et ainsi de pouvoir constater si la perte de linéaire séparabilité est oui ou non un phénomène qui va freiner l'efficacité de notre premier algorithme.



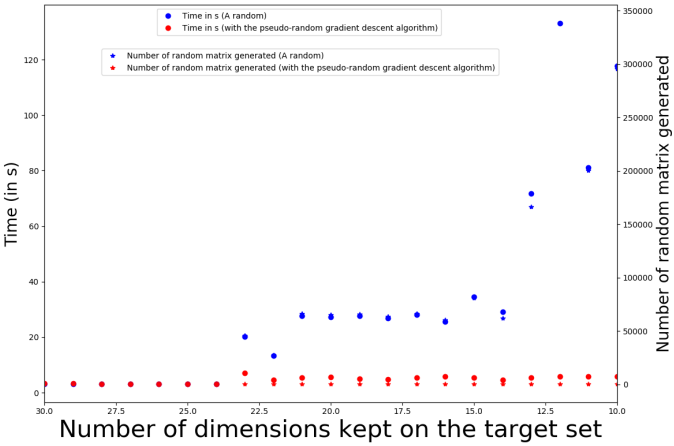

La nécessité d'un algorithme traitant des cas où on s'éloigne du cas linéairement séparable devient alors nécessaire :

## 3.4 Algorithme pseudo-random de descente de gradient :

— On génère aléatoirement une matrice, puis par sauts successifs de valeurs de ses coefficients on essaie de réduire l'erreur sur le set cible, si on ne parvient pas à passer le seuil voulu rapidement, on essaie avec une nouvelle matrice

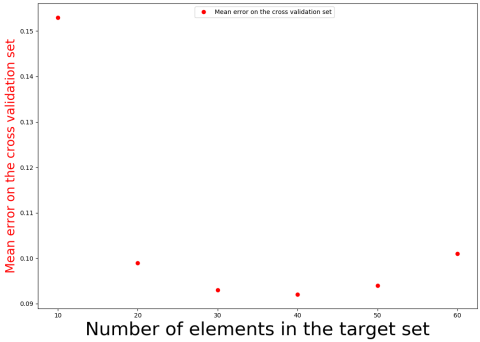
— Une fois parvenue à une bonne matrice, on en génère d'autres autour de cette dernière, ce qui nous donne plusieurs classifieurs que l'on booste via AdaBoost pour obtenir un classifieur $H_T$

Voir en annexe [A] le détail de l'algorithme en question plutôt complexe qui donne un moyen rapide de générer des sauts de gradient. Ici encore une fois, on fait le pari d'inclure dans les classificateurs à renforcer un bon classificateur niveau risque de manière à avoir un classificateur boosté encore meilleur que si tout avait été déterminé au hasard.




## 3.5 Etude de la première application en fonction du nombre d'éléments dans le set cible

Comme nous l'observons sur la courbe ci-dessous, à partir d'un certain seuil on peut voir un phénomène d'**overfitting** qui pourrait être compensé par une meilleure connaissance de la relation liant le domaine source et le domaine cible.

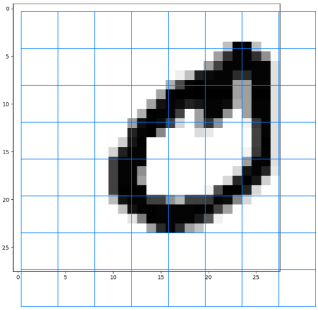


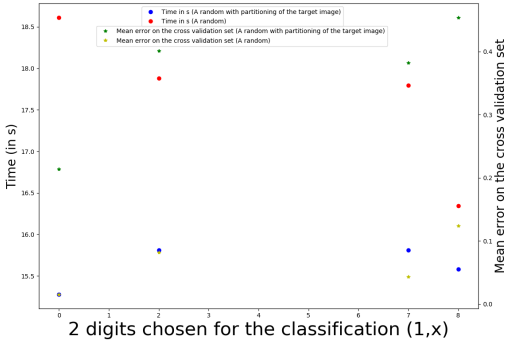

## 3.6 Deuxième application : reconnaissance de chiffres

Dans cette seconde application, nos 2 domaines sont des chiffres écrits à la main dont on a une image en 8x8 pixels en source, et en 28x28 pixels en cible.

Un point intéressant qu'on va souligner ici est qu'un choix aléatoire de matrice A va permettre de meilleurs résultats qu'en paramétrant A avec l'intuition qu'on peut se faire du passage du domaines 28x28 au domaine 8x8.

En effet, au lieu de randomiser tous les coefficients de A, la logique voudrait qu'on ne randomise que ceux qui permettront de transformer l'image 28x28 en image 7x7 (en laissant nuls les coefficients en trop pour l'image 8x8) avec des carrés de 4x4 pixels comme ci-dessous :






Comme on peut le voir sur le benchmark ci-dessus réalisé sur 100 loops, il est clair que dans le cas où tous les coefficients sont aléatoires on a de meilleur résultat. Cela peut s'expliquer en partie par le fait que les chiffres ne sont pas toujours centré de la même manière.

Un point important soulevé ici est alors la puissance de l'approche avec AdaBoost qui va permettre d'obtenir des résultats satisfaisants tout en ne percevant pas nécessairement tous les liens entre les 2 domaines.

## 3.7 Quelques éléments clés et exploration futures

- Le phénomène à éviter : transfert négatif que l'on n'a pas rencontré en utilisant AdaBoost en sortie même si ce n'est pas une garantie surtout en terme d'overfitting visible lors de la phase de **cross-validation**.
- L'effcacité du transfert en supposant les données sources et cible linéairement séparables.
- Lorsque l'on est plus dans un cas linéairement séparable, il devient rapidement plus difficile de trouver des matrices de projections appropriées pour un seuil fixé, cependant l'algorithme pseudo-random de descente par gradient, et l'augmentation du nombre de projections fournies à AdaBoost permettent de compenser ces phénomènes.
- Pour approfondir
  - Utiliser SVM (Support Vector Machine) au lieu de perceptron avec **l'astuce du noyau** consistant à se ramener à un espace de plus grande dimension pour le domaine où s'applique la classification et à transposer nos données et leurs labels via une fonction

"noyau", ce afin de se ramener du mieux possible à une séparatrice linéaire pour obtenir nos $X_{S0}$ et $X_{T0}$

## 3.8 Bibliographie

[1]Simon S. Du, Jayanth Koushik, Aarti Singh, Barnábas Poczos Hypothesis Transfer Learning via Transformation functions Carnegie Mellon University, 2017.

[2]Trevor Hastie, Robert Tibshirani, Jerome Friedman
*The Elements of Statistical Learning : Data Mining, Inference, and Prediction*
Stanford University, *Springer, 2009*

Annexe A

# Annexes

## A.1 Explication plus détaillée d'AdaBoost

### A.1.1 Principe de l'algorithme

Introduit par **Yoav Freund** and **Robert Schapire** qui ont gagné le prix Gôdel en 2003, il s'agit d'un algorithme de boosting[2] qui peut être combiné avec d'autres techniques d'apprentissage.

**Input :**

- A training set $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$.

**Initialization :**

- Maximum number of iterations $T$;
- initialize the weight distribution $\forall i \in \{1, \ldots, m\}, D^{(1)}(i) = \frac{1}{m}$.

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

- Learn a classifier $f_t : \mathbb{R}^d \to \{-1, +1\}$ using distribution $D^{(t)}$
- Set $\epsilon_t = \sum_{i : f_t(\mathbf{x}_i) \neq y_i} D^{(t)}(i)$
- Choose $a_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
- Update the weight distribution over examples

$$\forall i \in \{1, \ldots, m\}, D^{(t+1)}(i) = \frac{D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}}{Z^{(t)}}$$

where $Z^{(t)} = \sum_{i=1}^{m} D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}$ is a normalization factor such that $D^{(t+1)}$ remains a distribution.

**Output :** The voted classifier $\forall \mathbf{x}, F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} a_t f_t(\mathbf{x})\right)$

AdaBoost attribue à chaque classificateur en entrée un poids qu'il met à jour de la manière suivante. Dans la boucle, à l'itération courante(j-ième classificateur), il augmente ou diminue le poids de la i-ème donnée via l'exponentielle selon que la classification corresponde au label ou non. De ce fait, à la prochaine itération, cette donnée prendra plus d'importance si mal classée, moins sinon dans la fonction erreur du prochain meilleur classificateur au sens de la j+1-ième erreur(ou risque). Le prochain classificateur classera mieux cette donnée et entrera en compte dans la somme pondérée des classificateurs. Il est important d'avoir une liste de classificateurs qui classe mieux que le hasard(risque de 0.5), car sinon le processus de pondération aura l'effet inverse comme on peut le voir avec le pseudo-code. Pour se ramener à un classificateur à 2 classes, on prend le signe de l'expression à la fin de la boucle.

AdaBoost se sert en quelque sorte au fur et à mesure des forces et des uns et des autres et compense les faiblesses au fur et à mesure.

## A.2 Utilisation optimale d'AdaBoost

### A.2.1 Notamment dans les choix de l'estimateur source : les decision stumps

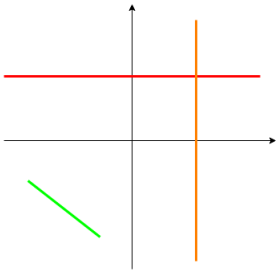
On s'est appuyé sur l'utilisation de AdaBoost, afin de booster des classifieurs faibles : **les decision stumps**

**Les decisions stumps :** Cas particuliers des arbres de décisions, les decision stumps sont définis par :

classe -1 : $x_i < b$

classe 1 : $x_i > b$

où $b \in \mathbb{R}$ et $i \in [|1, n|]$



En fait cette forme de classificateurs qui stratifie l'espace (le quadrille en 2D) une des plus utilisée dans le cas d'AdaBoost

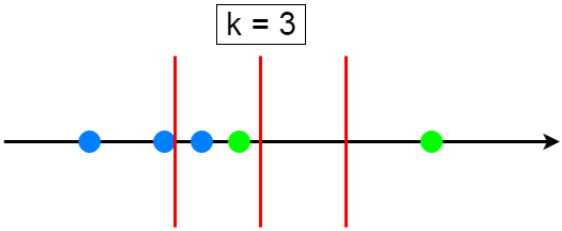
Un algorithme permettant de limiter le **fléau des dimensions** a été implémenté :

**La division de l'espace :** Dans chacune des dimensions i on divise la dimension du domaine en k parties entre le $min_i$ et $max_i$ avec :

$$min_i = \min_{X \in D_S} x_i \tag{A.1}$$

$$max_i = \max_{X \in D_S} x_i \tag{A.2}$$

En effet, lorsqu'on étudie de la classification où les données vivent dans un espace vectoriel de dimension $d$, on peut montrer que l'indétermination sur la position des points(données) dans l'espace en fonction de sa dimension $d$ grande évolue exponentiellement en $O(\zeta^d)$. Or, avec des classification de type decision stumps, on se ramène à des erreurs de l'ordre $d\zeta$ où $\zeta$ est l'erreur que l'on peut avoir sur $\mathbb{R}$.




## A.3 Un pseudo-code l'algorithme pseudo random de descente de gradient

C'est une version analogue au recuit simulé (*simulated annealing*). On cherche à ne pas être bloqué dans des minima locaux en permettant au

coefficient des matrices générées de faire un saut en terme de norme. Le détail se trouve dans le code fourni, et montre aussi que les méthodes utilisées sont encore au stade de recherche. Nous espérons pouvoir fournir un algorithme plus abouti dans les temps à venir.

## A.4 Démonstrations mathématiques en tout genre

3.2.2 (p11) :

Montrons que pour H et G 2 hyperplans de $\mathcal{R}^N$ et $\mathcal{R}^P$, avec $N \geq P$, il existe $\Psi \in \mathcal{L}(\mathcal{R}^N, \mathcal{R}^P)$ tel que $\Psi(H) = G$

**Démonstration :** Soit $e_N$ le vecteur normal pris de norme 1 à H dans $\mathcal{R}^N$ et $u_P$ vecteur normal dans $\mathcal{R}^P$. Soit $\zeta$ défini par : $\zeta =< e_N | u_P > e_N - u_P$ et normalisons-le. Complétons la famille orthonormée $(e_N, \zeta)$ en une base orthonormée, via le procédé de Gram-Schmidt Cette base $e$ contient alors des vecteurs de base de G, d'où en posant : $\Psi(e_i) = e_i$ pour tout $i \in \{1, \ldots, P-1\}$ et le reste à 0, on a notre application linéaire, et donc l'existence d'une matrice $A$ qui transforme l'hyperplan H en G.