

Synthèse Apprentissage inductif appliqué

Jean-Philippe Collette

4 janvier 2013

Table des matières

1	Introduction	4
1.1	Glossaire	4
1.2	Protocoles d'apprentissage automatique	5
1.2.1	Algorithme d'apprentissage	5
1.2.2	Apprentissage supervisé	5
1.2.3	Apprentissage non-supervisé	5
1.2.4	Apprentissage semi-supervisé	5
1.2.5	Apprentissage transductif	6
1.2.6	Apprentissage par renforcement	6
1.2.7	Apprentissage actif	7
1.3	Comparaison des méthodes d'apprentissage supervisé	7
I	Apprentissage supervisé	9
2	Arbres de décision	10
2.1	Arbre de classification	10
2.1.1	Création d'un arbre de classification	10
2.1.2	Recherche du meilleur attribut	10
2.1.3	Élagage	11
2.1.4	Variables numériques	12
2.1.5	Attributs à plusieurs valeurs	12
2.1.6	Valeurs d'attribut manquantes	13
2.2	Arbre de régression	13
2.3	Interprétabilité et sélection d'attribut	13
2.4	Avantages et inconvénients	14
2.5	Noyau associé à un arbre	14
3	<i>k</i>-NN - méthode des <i>k</i>ème plus proches voisins	15
3.1	Algorithme du plus proche voisin	15
3.2	Propriétés	15
3.3	Raffinements	16
3.3.1	La méthode <i>k</i> -NN	16
3.3.2	Condensation et édition de LS	16
3.3.3	Autres raffinements	17
3.4	Avantages et inconvénients	17
3.5	Relation entre les méthodes à base d'arbres et à base de noyaux	17
3.6	Relation entre les méthodes linéaires et à base de noyaux	18
4	Méthodes linéaires	19
4.1	Introduction	19
4.2	Ridge regression	19
4.3	Perceptron	21
4.3.1	Algorithme du perceptron	21
4.3.2	Soft threshold units	21
4.3.3	Descente de gradient	22
4.3.4	Propriétés	22
4.3.5	Couche de perceptrons	22
4.4	Réseaux de neurones	22

4.4.1	Classification	22
4.4.2	Régression	23
4.4.3	Apprentissage d'un réseau multi-couche	23
4.5	Méthodes de régression à base de noyaux	23
5	Biais et variance	25
5.1	Problème de régression sans variable d'entrée	25
5.2	Décomposition biais/variance	26
5.2.1	Approche bayesienne	28
5.2.2	Problème de régression (complet)	28
5.2.3	Illustration	28
5.2.4	Lien entre biais/variance et sous/sur-apprentissage	29
5.3	Problèmes de classification	29
5.4	Paramètres influençant le biais et la variance	30
5.4.1	Complexité du modèle	30
5.4.2	Bruit	31
5.4.3	Taille de l'échantillon d'apprentissage	32
5.4.4	Algorithmes d'apprentissage	32
5.5	Techniques de réduction du biais et de la variance	32
5.5.1	Réduction de la variance	33
6	Évaluation de modèles	34
6.1	Erreurs	34
6.1.1	Erreur de resubstitution	34
6.1.2	Erreur de généralisation	34
6.2	Méthodes d'évaluation	34
6.2.1	Méthode test set	34
6.2.2	K-fold	35
6.2.3	Impact de la complexité d'un modèle avec une validation croisée	36
6.2.4	Bootstrap	36
6.2.5	Erreurs de test conditionnelles et erreurs de test attendues	37
6.3	Méthodes de sélection	37
6.3.1	Méthode test set	37
6.3.2	K-fold	37
6.3.3	Méthodes analytiques	38
6.4	Biais de sélection	38
6.5	Mesure de performance	38
6.5.1	Classification binaire	38
6.5.2	Régression	40
6.5.3	Mesures de performances pour l'entraînement	41
7	Machines à support vectoriel	42
7.1	Machines à support vectoriel linéaire	42
7.1.1	Hyperplan de marge maximale	42
7.1.2	Problème d'optimisation	43
7.1.3	Vecteurs de support	44
7.1.4	Rappel du lagrangien	45
7.1.5	Soft margin	45
7.2	Noyaux dans les SVMs	47
7.3	Kernel trick	48
7.4	Notion mathématique du noyau	48
7.5	Exemple de noyaux	48
7.6	Méthodes à noyaux	49
7.7	Forces et faiblesses des SVMs	49

8 Méthodes d'ensemble	50
8.1 Bagging	50
8.1.1 Approche théorique	50
8.1.2 En pratique	50
8.1.3 Autres techniques de moyennage	52
8.2 Random Forests	52
8.3 Décomposition de l'ambiguïté	52
8.4 Boosting	53
8.4.1 Adaboost	54
8.4.2 Boosting aux moindres carrés	54
8.4.3 Algorithme de boosting générique	55
8.4.4 LS boosting	55
8.4.5 Autres méthodes de boosting	56
8.5 Interprétabilité et efficacité	56
8.6 Autres approches d'ensemble	56
8.6.1 Moyenne de modèles de Bayes	56
8.6.2 Stacking	56
8.7 Conclusion	57
9 Sélection de features	58
9.1 Techniques de filtre	58
9.2 Techniques embarquées	58
9.2.1 LASSO	59
9.3 Techniques de wrapper	59
9.4 Biais de sélection	60
II Apprentissage non supervisé	61
10 Apprentissage non supervisé	62
10.1 Clustering	62
10.1.1 Mesure de distances	63
10.1.2 Clustering hiérarchique	63
10.1.3 K-means	65
10.1.4 Self-Organizing Maps	67
10.1.5 Nombre de cluster	68
10.1.6 Sélection de features	68
10.2 Réduction de dimensionnalité	68
10.2.1 PCA	69
10.2.2 Approche mathématique	69
10.2.3 Etude des composantes	71
10.2.4 Limitations	72
10.2.5 Extensions de PCA	72
10.2.6 Autres techniques de réduction de dimension	72
10.3 Autres méthodes non-supervisées	73

Chapitre 1

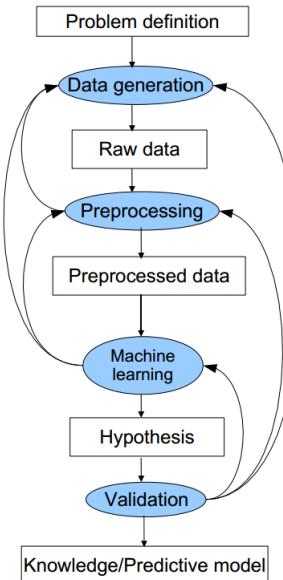
Introduction

L'apprentissage consiste à

- améliorer les performances d'un ordinateur dans certaines tâches avec de l'expérience ;
- extraire un modèle d'un système en se basant sur les observations de ce système dans certaines situations ;
- créer un modèle, c'est-à-dire une relation entre les variables utilisées pour décrire le système.

Les deux buts principaux de l'apprentissage sont la prédition et la meilleure compréhension d'un système.

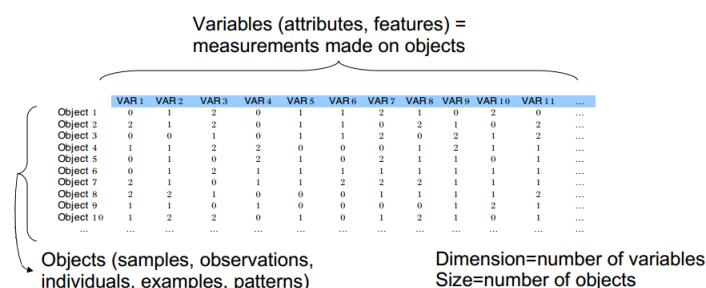
L'apprentissage est utilisé quand il n'y a pas d'expertise humaine, quand les humains ne sont pas capables d'expliquer leur expertise, quand les solutions changent au cours du temps ou quand les solutions nécessitent d'être adaptées à des cas particuliers.



L'exploration de données se déroule en plusieurs étapes :

1. Génération de données ;
2. Préprocessing : normalisation des valeurs, traitement des valeurs manquantes, sélection de composantes, etc ;
3. Apprentissage : développement d'une hypothèse, choix d'un algorithme d'apprentissage, etc ;
4. Validation d'hypothèse : validation croisée, déploiement du modèle, etc.

1.1 Glossaire



1.2 Protocoles d'apprentissage automatique

1.2.1 Algorithme d'apprentissage

Un algorithme d'apprentissage est défini par

- une famille de modèles candidats (un espace d'hypothèses H),
- une mesure de la qualité d'un modèle, et
- une stratégie d'optimisation.

L'algorithme va ainsi, à partir de l'échantillon d'apprentissage, retourner la fonction h de H de meilleur qualité.

Modes batch et on-line

En mode batch, les échantillons sont fournis et traités ensembles pour construire un modèle. Il n'est pas nécessaire de stocker le modèle mais bien ces exemples. C'est une approche classique pour l'exploitation de données (data mining).

En mode online, les échantillons sont fournis et traités un à un pour mettre à jour un modèle. Ici, il faut stocker le modèle (et non les exemples). C'est une approche fortement utilisée pour les systèmes adaptatifs.

On peut cependant passer facilement d'un mode à l'autre :

- les échantillons peuvent être fournis ensembles, mais exploités un à un
- les échantillons peuvent être fournis un à un, mais sont stockés et exploités ensemble.

1.2.2 Apprentissage supervisé

L'apprentissage supervisé consiste, à partir d'une base de données (learning sample, échantillon de test), à trouver une fonction f qui prend en entrée les variables du problème et qui approxime au mieux la sortie :

$$\hat{Y} = f(X_1, X_2, X_3, X_4)$$

Plus formellement, l'apprentissage consiste, à partir d'un échantillon d'apprentissage $\{(x_i, y_i) | i = 1, \dots, N\}$, avec $x_i \in \mathcal{X}$ et $y_i \in \mathcal{Y}$, à trouver une fonction $f : \mathcal{X} \rightarrow \mathcal{Y}$ qui minimise la fonction de probabilité de perte $l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ sur la distribution jointe des paires d'entrées-sortie : $E_{x,y}\{l(f(x), y)\}$.

Cette fonction de perte l prend en entrée deux sorties Y et retourne 1 si elles sont équivalentes, 0 sinon.

Lorsque la sortie est une valeur symbolique, on parle de classification. Si la sortie est une valeur numérique, on parle de régression.

Un modèle sera déterministe s'il est parfait, c'est-à-dire s'il a une règle de classification qui ne commet pas d'erreur.

Exemples d'application

- reconnaissance/labellisation d'images
- détecteur d'injures
- système de recommandation de produits sur un site de vente

1.2.3 Apprentissage non-supervisé

L'apprentissage non-supervisé consiste à trouver des régularités dans les données, sans indications sur des entrées et des sorties. On cherche ainsi quels sont les groupes de variables ou d'objets intéressants, et s'il y a des dépendances entre les variables.

Exemples d'application

- classification de documents sur le web
- étude de tremblements de terre
- classification de plantes et d'animaux selon leurs caractéristiques
- marketing : trouver des groupes de clients avec des comportements similaires

1.2.4 Apprentissage semi-supervisé

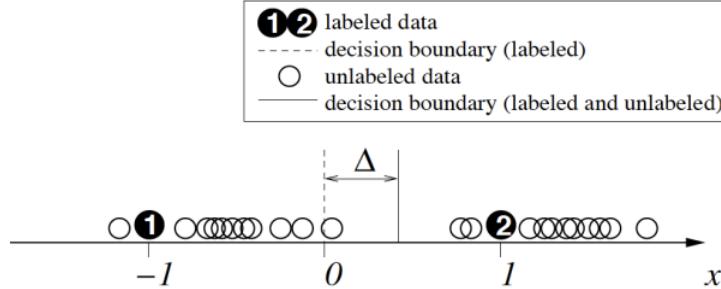
Il faut distinguer deux types de données : celles qui sont labellisées de celles qui ne le sont pas. Une donnée non labellisée est une paire entrée-sortie sans valeur de sortie. Elles sont plus faciles à obtenir que des données labellisées car

- les labels peuvent nécessiter une intervention/expertise humaine, qui peut être chère, lente et non fiable
- les labels peuvent nécessiter des équipements spéciaux.

Des données non labellisées se retrouvent dans le domaine biomédical, dans l'analyse du langage, dans le parсage du langage naturel, dans la catégorisation d'images ou dans la mesure de réseaux.

Le but de l'apprentissage semi-supervisé est d'exploiter tout type de données, labellisées ou non, et de construire de meilleurs modèles pas qu'en utilisant seulement un seul de ces types.

On a un gain potentiel si on considère les données non labellisées.



L'apprentissage semi-supervisé se fait selon deux méthodes :

- de l'auto-apprentissage : on labellise les données non labellisées avec un modèle entraîné sur les données labellisées ;
- des algorithmes semi-supervisés, par exemple le semi-supervised SVM : on énumère tous les labels possibles pour les données non labelisées, on apprend une SVM pour chaque labeling, et on prend celui qui donne la plus grande marge.

Exemples d'application

Les problèmes d'apprentissage naturels sont résolus avec des méthodes proches de l'apprentissage semi-supervisé.

Par exemple, on peut apprendre à un enfant ce que sont un chien et un chat en lui présentant un chien et un chat d'une race particulière. Lorsqu'il sera confronté à des chiens d'autres races (des données non labellisées), il saura les différencier des chats d'autres races, même s'il ne les a jamais vu.

1.2.5 Apprentissage transductif

Ce type d'apprentissage est similaire à l'apprentissage supervisé, mais on a accès aux données de test dès le début, ce qu'on exploite plus. On ne veut pas un modèle, on veut calculer des prédictions pour des données non labellisées.

Pour cela, on utilise de l'apprentissage semi-supervisé en utilisant les données à tester comme données non labellisées pour obtenir un modèle, et on l'utilise pour faire des prédictions sur les données de test. Il existe également des algorithmes spécifiques qui évitent purement et simplement la construction d'un modèle.

A l'opposé, l'apprentissage inductif a pour but de trouver la règle pour associer la sortie à des attributs.

Algorithme : partitionning transduction

On considère un ensemble contenant tous les points, sous une seule grande partition.

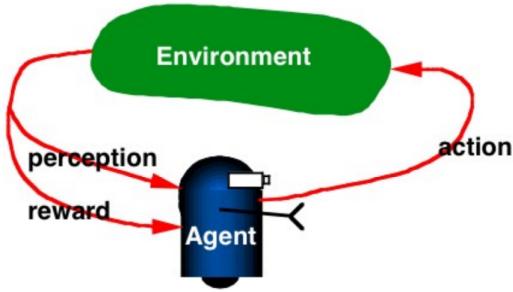
- tant qu'au moins une partition P contient deux points de labels différents :
 - partitionner P en des plus petites partitions
 - pour chaque partition P :
 - assigner le même label à tous les points de P

1.2.6 Apprentissage par renforcement

Il s'agit d'un apprentissage basé sur des interactions :

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Le but est de choisir une séquence d'actions (une politique) qui maximise $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$, avec $0 \leq \gamma < 1$.



Un système est généralement modélisé par

- des probabilités de transition à un état $P(s_{t+1}|s_t, a_t)$
- des probabilités de récompense $P(r_{t+1}|s_t, a_t)$ (processus de décision de Markov).

Si le modèle des dynamiques et de la récompense est connu, on va calculer la politique optimale à adopter par programmation dynamique.

Si le modèle n'est pas connu, deux approches sont possibles :

- on se base sur un modèle : on apprend d'abord un modèle des dynamiques et on en dérive une politique optimale
- on n'utilise pas de modèle : on apprend directement la politique en se basant sur les observations des trajectoires du système.

Renforcement vs. supervisé

	Supervisé	Renforcement
Mode batch	Apprentissage d'un mapping d'une entrée vers une sortie à partir d'observations de paires entrée-sortie	Apprentissage d'un mapping d'un état vers une action à partir de triplets (état, action, récompense) observés
Mode online	(Active learning) combinaison d'apprentissage supervisé et de sélection (online) d'instances pour labeller	Combinaison d'apprentissage de politique avec un contrôle du système et avec la génération de trajectoires d'entraînement

L'apprentissage par renforcement pourrait se réduire à de l'apprentissage supervisé si l'action optimale était connue pour chaque état. De plus, l'apprentissage supervisé est utilisé par l'apprentissage par renforcement pour modéliser les dynamiques du système et/ou les fonctions de récompense.

Exemples d'application

- robotique
- agent dans un jeu

1.2.7 Apprentissage actif

Pour des données non labellisées, on veut trouver (de manière adaptative) des exemples à labelliser afin d'apprendre un modèle précis. L'espoir est de réduire le nombre d'instances labellisées en utilisant l'apprentissage supervisé de type batch.

Généralement, on a une configuration de type online : on choisit les k meilleurs exemples non labellisés, on détermine leurs labels, on met à jour le modèle et on itère.

Les algorithmes diffèrent dans la façon dont les exemples non labellisés sont choisis. Des adaptations de SVMs existent et des perceptrons sont utilisés pour de l'apprentissage actif.

1.3 Comparaison des méthodes d'apprentissage supervisé

Method	Accuracy	Efficiency	Interpretability	Ease of use
kNN	++	+	+	++
DT	+	+++	+++	+++
Linear	++	+++	++	+++
Ensemble	+++	+++	++	+++
ANN	+++	+	+	++
SVM	++++	+	+	+

A noter que l'importance relative des critères dépend de l'application, et que ce ne sont que des tendances générales.

Première partie

Apprentissage supervisé

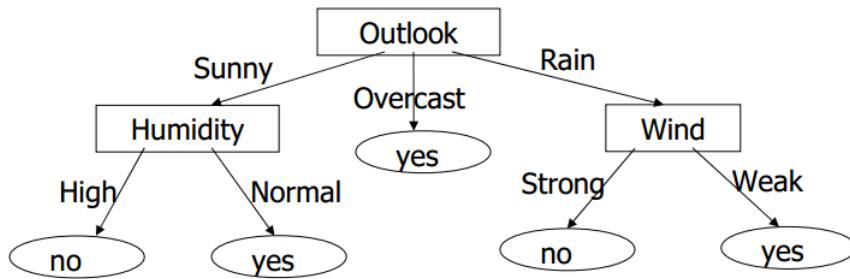
Chapitre 2

Arbres de décision

Il s'agit d'un algorithme d'apprentissage qui peut gérer les problèmes de classification (binaire ou avec plusieurs valeurs) et avec des attributs qui peuvent être discrets ou continus.

Un arbre de décision est un arbre où

- chaque noeud intérieur teste un attribut,
- chaque branche correspond à la valeur d'un attribut, et
- chaque feuille est labellisée par une classe.



L'apprentissage avec cet algorithme consiste à choisir la structure d'arbre et à déterminer les prédictions aux feuilles.

Afin de minimiser l'erreur de classification, on associe aux feuilles la classe majoritaire lorsqu'on descend dans l'arbre.

2.1 Arbre de classification

2.1.1 Création d'un arbre de classification

On a l'algorithme suivant (Top-down induction of DT), pour une procédure $learn_dt(LS)$, où LS est l'échantillon d'apprentissage.

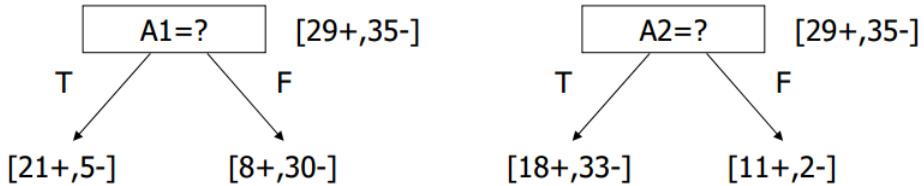
- si tous les objets de LS ont la même classe, créer une feuille avec cette classe comme label,
- sinon,
 - trouver le meilleur attribut A pour une séparation,
 - créer un noeud de test pour cet attribut, et
 - pour chaque valeur a de A ,
 - construire $LS_a = \{o \in LS | A(o) = a\}$, et
 - utiliser $learn_dt(LS_a)$, pour créer un sous-arbre à partir de LS_a .

Propriétés :

- sous-optimal mais rapide
- dépend très fortement du critère pour sélectionner l'attribut

2.1.2 Recherche du meilleur attribut

Pour trouver le meilleur attribut, il faut définir un score afin d'évaluer les séparations possibles. Ce score devra favoriser la séparation en classes, afin de réduire la profondeur de l'arbre.



Impureté

Une mesure assez commune est l'impureté. Soit p_j la proportion d'objets de la classe j ($j = 1, \dots, J$) dans LS.

– $I(LS)$ est minimal si $p_i = 1$ et $p_j = 0$ pour $j \neq i$.

– $I(LS)$ est maximal si $p_j = \frac{1}{J}$: il y a le même nombre d'objets de chaque classe.

Le meilleur split est celui qui maximise la réduction d'impureté.

$$\Delta I(LS, A) = I(LS) - \sum_a \frac{|LS_a|}{|LS|} I(LS_a)$$

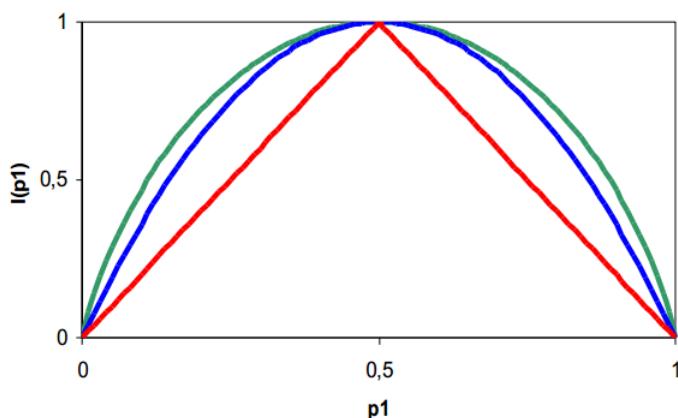
avec LS_a le sous-ensemble des objets de LS tels que $A = a$. ΔI est une mesure de score ou un critère de séparation.

Exemple de mesure d'impureté :

– **entropie de Shannon** : $H(LS) = \sum_j p_j \log_2 p_j$. Elle mesure l'incertitude, la surprise.

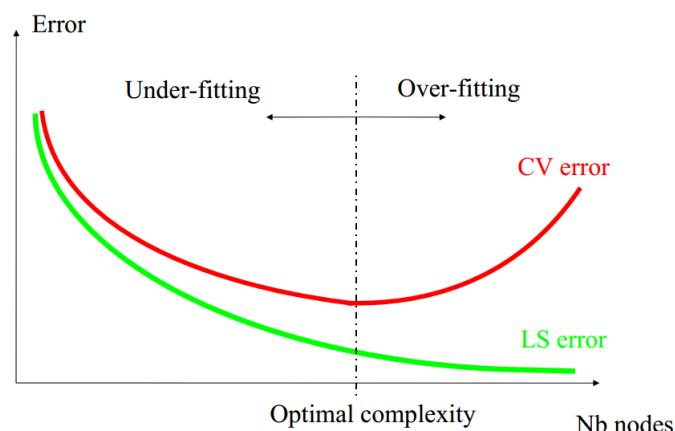
– **indice de Gini** : $I(LS) = \sum_j p_j(1 - p_j)$

– **taux d'erreur de mal-classification** : $I(LS) = 1 - \max_j p_j$. En pratique, ce n'est jamais utilisé car cette mesure ne fait pas la différence entre des situations très différentes. Par exemple, une classification où on passe d'un score de 90% à 100% est bien plus intéressante qu'une classification qui passe de 50% à 60%, pourtant la différence d'impureté sera la même.



2.1.3 Élagage

Le sur-apprentissage survient lorsqu'on considère un arbre trop profond et qu'on tient compte de bruit ou de mauvaises mesures dans les données.



Pour éviter l'overfitting, on a trois méthodes :

– pre-pruning/pré-élagage : arrêter d'étendre l'arbre plus tôt, avant qu'il n'atteigne le point où il classe parfaitement l'échantillon d'apprentissage ;

- post-pruning/post-élagage : permettre à l'arbre de sur-apprendre et de l'élaguer ensuite ;
- les méthodes d'ensemble.

Pre-pruning

On arrête la séparation de noeud si

- le nombre d'objets est trop petit (*min_samples_split*), ou si
- l'impureté est trop faible, ou si
- le meilleur test n'est pas statistiquement pertinent

Le problème est que le paramètre optimal pour cet élagage est très dépendant du problème, et que l'on peut passer à côté de l'arbre optimal.

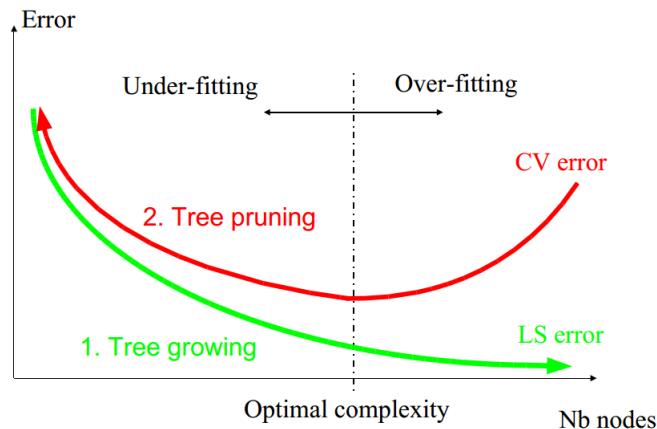
Post-pruning

On sépare l'ensemble d'apprentissage LS en deux :

- un ensemble GS (growing sample) pour construire l'arbre, et
- un ensemble de validation VS pour évaluer l'erreur de généralisation.

On va construire un arbre complet à partir de GS, puis on va calculer un ensemble d'arbres $\{T_1, T_2, \dots\}$, avec T_1 l'arbre complet et T_i l'arbre obtenu en retirant des noeuds de l'arbre T_{i-1} .

On sélectionne ensuite le T_{i^*} de la séquence qui minimise l'erreur sur VS.



Pour construire la séquence, on a plusieurs manières :

- reduced error pruning : à chaque étape, on retire le noeud qui diminue le plus l'erreur sur VS
- cost-complexity pruning : on définit un critère de coût-complexité :

$$\text{Error}_{VS}(T) + \alpha \text{ Complexity}(T)$$

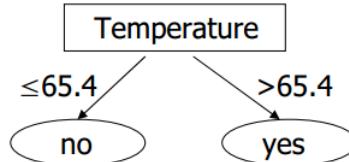
On construit la séquence d'arbres qui minimise ce critère en augmentant α .

Le problème du post-pruning est qu'il faut dédier une partie de l'ensemble d'apprentissage à la validation, ce qui peut poser un problème pour des petites bases de données. La solution est d'utiliser de la validation croisée k -fold.

2.1.4 Variables numériques

Deux solutions :

- pré-discretiser, assigner des valeurs symboliques à des ranges (par exemple "froid" si la température est inférieure à 70°F , "normal" si entre 70 et 75°F , "chaud" si plus de 75°F) ;
- discréteriser durant l'opération de construction de l'arbre, en prenant le seuil qui maximise l'impureté.



2.1.5 Attributs à plusieurs valeurs

Effectuer un split sur un attribut à plusieurs valeur n'est pas une bonne chose car il y a trop de fragmentation et la réduction d'impureté est souvent trop grande. Dès lors, on peut

- n'autoriser que les splits binaires
- dans le critère de choix du meilleur attribut, pénaliser les valeurs multiples

2.1.6 Valeurs d'attribut manquantes

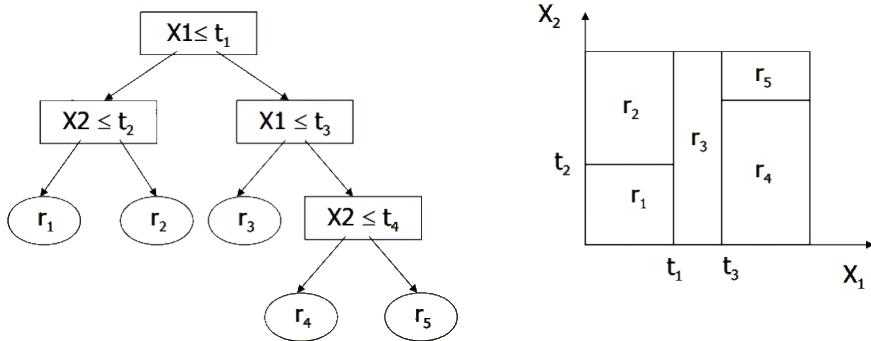
On peut soit

- assigner la valeur la plus fréquente dans LS
- assigner la valeur la plus fréquente dans l'arbre
- assigner une probabilité à chaque valeur possible

Slides 38 → 44 : variables numériques, attributs à plusieurs valeurs, valeurs d'attribut manquantes.

2.2 Arbre de régression

Un arbre de régression est un arbre de décision où les labels des noeuds sont numériques.



Afin de minimiser l'erreur quadratique sur LS, la prédiction à une feuille est la moyenne de tous les éléments de LS qui l'atteignent.

L'impureté d'un échantillon est définie par la variance de la sortie de cet échantillon :

$$I(LS) = \text{var}_{y|LS}\{y\} = E_{y|LS}\{(y - E_{y|LS}\{y\})^2\}$$

Le meilleur split est celui qui diminue le plus la variance :

$$\Delta I(LS, A) = \text{var}_{y|LS}\{y\} - \sum_a \frac{|LS_a|}{|LS|} \text{var}_{y|LS_a}\{y\}$$

L'élagage fonctionne de la même façon que pour les arbres de classification, si ce n'est que dans le post-pruning, c'est l'arbre qui minimise l'erreur quadratique sur VS qui est sélectionné.

En pratique, le pruning est plus important en régression car les arbres complets sont beaucoup plus complexes ; souvent, tous les objets ont une sortie différente, du coup l'arbre complet a autant de feuilles qu'il y a d'objets dans l'échantillon d'apprentissage.

2.3 Interprétabilité et sélection d'attribut

Un arbre de décision est très interprétable, il peut être converti facilement en un ensemble de règles "si ... alors".

Si certains attributs ne sont pas nécessaires pour la classification, il n'apparaîtront pas dans l'arbre (élagé/pruned). C'est important si la mesure de certaines variables est coûteuse.

Les arbres de décision sont souvent utilisés comme pre-processing pour d'autres algorithmes d'apprentissage, qui souffrent de variables inutiles.

Certaines variables ont une importance, elles ne contribuent pas toutes de manière égale. Grâce aux arbres, on peut évaluer leur importance. Cela peut permettre notamment d'éviter de mesurer des attributs si cela est coûteux et inutile.

L'importance d'une variable est calculée par la réduction de l'impureté :

$$I(A) = \sum_{\substack{\text{noeuds où} \\ A \text{ est testé}}} |LS_{\text{noeud}}| \Delta I(LS_{\text{noeud}}, A)$$

2.4 Avantages et inconvénients

- + très rapide et scalable, on peut traiter d'énorme quantité d'entrées et d'objets ; la complexité est de l'ordre $\mathcal{O}(nN \log N)$
- + donne une bonne interprétabilité et quantifie l'importance des variables ;
- + très flexible : support de plusieurs types d'attribut, de valeurs manquantes, de problèmes de classification et de régression, etc
- grande variance et donc grande instabilité ;
- souvent pas aussi précis que d'autres méthodes.

2.5 Noyau associé à un arbre

On peut définir un noyau facilement : $K_t(x, x') = 1$ si, lorsqu'on parcourt l'arbre t , x et x' arrivent dans la même feuille, 0 sinon.

Le modèle pour un arbre t est ainsi

$$f_t(x) = \sum_{i=1}^N y(x_i) K_t(x_i, x)$$

Pour un ensemble T de M arbres, le noyau est égal à la moyenne du nombre de fois que les deux objets arrivent dans une même feuille :

$$K_T(x, x') = \frac{1}{M} \sum_{j=1}^M K_{t_j}(x, x')$$

Le modèle est alors

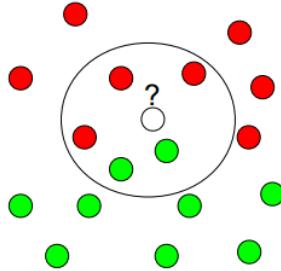
$$f_T(x) = \frac{1}{N} \sum_{i=1}^N y(x_i) K_T(x_i, x)$$

Chapitre 3

k -NN - méthode des k ème plus proches voisins

3.1 Algorithme du plus proche voisin

Cette méthode consiste à prédire la sortie en se basant sur les plus proches voisins de l'entrée. L'idée est que les objets similaires devraient avoir des sorties similaires.



Pour se faire, on trouve les k plus proches voisins en utilisant une mesure de la distance (si n est le nombre de features) :

$$d_a(o, o') = (\mathbf{a}(o) - \mathbf{a}(o'))^T (\mathbf{a}(o) - \mathbf{a}(o')) = \sum_{i=1}^n (a_i(o) - a_i(o'))^2$$

Le plus proche voisin est donné par

$$NN_a(o, LS) = \arg \min_{o' \in LS} d_a(o, o')$$

On extrapole alors la sortie sur base du plus proche voisin :

$$\hat{y}_{NN}(o) = y(NN_a(o, LS))$$

3.2 Propriétés

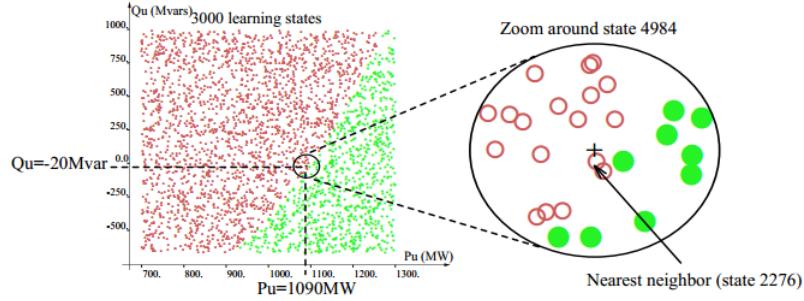
Au niveau de la complexité :

- pour l'apprentissage, il faut stocker LS : $n \times N$ (pour N objets de n features) ;
- pour les tests, il faut calculer N distances, on a donc $N \times n$ calculs à effectuer.

Au niveau de la précision, asymptotiquement (\Leftrightarrow lorsque $N \rightarrow \infty$), on est sous-optimal (sauf si le problème est déterministe).

De plus, il y a une forte dépendance sur le choix des attributs. On considère alors des poids sur les attributs lors du calcul de la distance :

$$d_a^w = \sum_{i=1}^n w_i (a_i(o) - a_i(o'))^2$$



L'erreur de resubstitution ne sera pas nulle si on considère les points sur la frontière. Ainsi, le point vert à côté des deux rouges sera mal classé si on considère les trois plus proches voisins.

3.3 Raffinements

3.3.1 La méthode k -NN

On considère les k plus proches voisins, ce qui donne

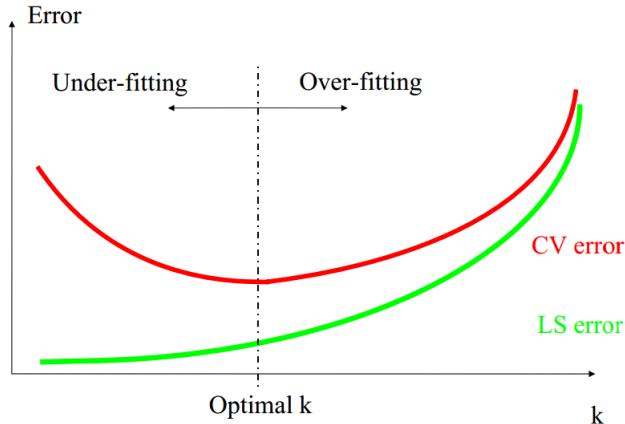
$$kNN(o, LS) = \text{First}(k, \text{sort}(LS, d_a(o, .)))$$

La sortie sera,

- dans le cadre d'une classification, la classe la plus fréquente,
- dans le cadre d'une régression, la valeur moyenne.

$$\hat{y}_{kNN}(o) = k^{-1} \sum_{o' \in kNN_a(o, LS)} y(o')$$

k permet de contrôler le sur-apprentissage. Asymptotiquement ($\Leftrightarrow N \rightarrow \infty$) : $k(N) \rightarrow \infty$ et $\frac{k(N)}{N} \rightarrow 0$, on a donc une solution optimale (l'erreur est minimisée).



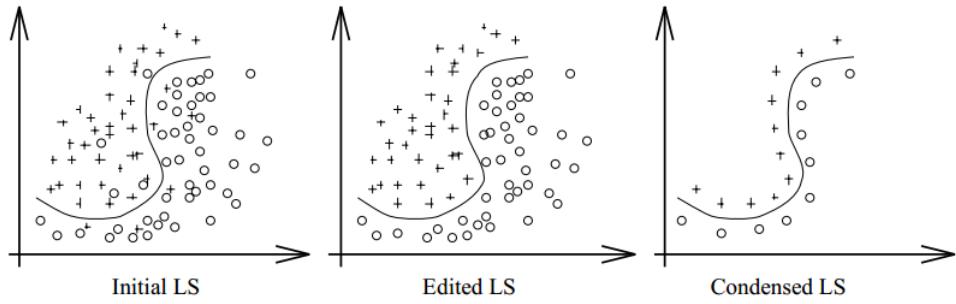
k ne doit pas augmenter trop vite avec N , par exemple $k = \sqrt{N}$.

3.3.2 Condensation et édition de LS

L'idée est

- de condenser LS, c'est-à-dire retirer les objets "inutiles",
- d'édition LS en retirant des valeurs aberrantes.

En appliquant ces deux modifications (d'abord l'édition puis la condensation), on élague LS afin d'obtenir de meilleures performances.



3.3.3 Autres raffinements

- mise au point automatique du vecteur de poids w
- utiliser des fenêtres de Parzen et/ou des méthodes à base de noyaux :

$$\hat{y}_K(o) = \sum_{o' \in LS} y(o')K(o, o')$$

où $K(o, o')$ est une mesure de la similarité. On pourrait par exemple prendre $K(o, o') = -\alpha||a(o) - a(o')||^2$: cela permet de faire intervenir plus les voisins proches et moins ceux qui sont éloignés (réglable via α).

3.4 Avantages et inconvénients

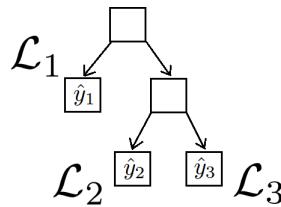
- + très simple ;
- + peut être adapté pour tout type de données, en changeant la mesure de la distance ;
- choisir une bonne mesure de la distance est un problème compliqué ;
- cet algorithme est très sensible à la présence de bruit ;
- lent car il faut calculer N distances par prédiction.

3.5 Relation entre les méthodes à base d'arbres et à base de noyaux

Un noyau peut être défini par un arbre de régression. Soient

- \mathcal{L}_i , $i = 1, \dots, |\mathcal{T}|$, l'ensemble des feuilles de \mathcal{T} ,
 - N_i le nombre d'objets dans le sous-LS de \mathcal{L}_i , et
 - $K_{\mathcal{T}}(o, o')$ égal à $\frac{1}{N_i}$ si o et o' atteignent la même feuille \mathcal{L}_j , 0 sinon.
- Alors l'approximation de l'arbre de régression peut être écrite comme

$$\hat{y}_{\mathcal{T}}(o) = \sum_{o' \in LS} y(o')K_{\mathcal{T}}(o, o')$$



Il s'agit de la moyenne des objets de LS qui se trouvent dans la feuille \mathcal{L}_i , la même feuille que celle dans laquelle l'objet o atterrit.

On peut utiliser une représentation à base de produits scalaires :

- pour chaque feuille, on définit une fonction $a_{\mathcal{L}_i}(o)$ par $a_{\mathcal{L}_i}(o) = N_i^{-\frac{1}{2}}$ si o atteint \mathcal{L}_i , 0 sinon.
- soit $\mathbf{a}_{\mathcal{T}}(o) = (a_{\mathcal{L}_1}(o), \dots, a_{\mathcal{L}_{|\mathcal{T}|}}(o))^T$

On a alors

$$K_{\mathcal{T}}(o, o') = \mathbf{a}_{\mathcal{T}}^T(o)\mathbf{a}_{\mathcal{T}}(o')$$

et

$$\hat{y}_{\mathcal{T}}(o) = \sum_{o' \in LS} y(o')\mathbf{a}_{\mathcal{T}}^T(o)\mathbf{a}_{\mathcal{T}}(o')$$

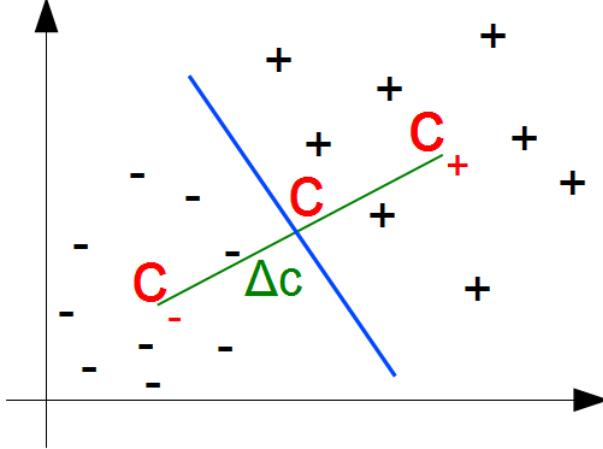
3.6 Relation entre les méthodes linéaires et à base de noyaux

On considère un problème de classification et on définit $y(o) = 1$ si $c(o) = c_1$ et $y(o) = -1$ si $c(o) = c_2$. On peut construire un

- le centre de la classe 1 : $\mathbf{c}_+ = N_+^{-1} \sum_{o' \in LS_+} \mathbf{a}(o')$
- le centre de la classe 2 : $\mathbf{c}_- = N_-^{-1} \sum_{o' \in LS_-} \mathbf{a}(o')$
- classificateur : $\hat{y}(o) = 1$ si $d(\mathbf{c}_+, \mathbf{a}(o)) < d(\mathbf{c}_-, \mathbf{a}(o))$, -1 sinon
- on définit $c = \frac{c_+ + c_-}{2}$ et $\Delta\mathbf{c} = \mathbf{c}_+ - \mathbf{c}_-$

On a alors

$$\hat{y}(o) = \text{sgn}((\mathbf{a}(o) - \mathbf{c})^T \Delta\mathbf{c})$$



En distribuant $\Delta\mathbf{c}$ et en substituant les valeurs de \mathbf{c} , \mathbf{c}_+ et \mathbf{c}_- , on a l'expression à base de produit scalaires

$$\hat{y}(o) = \text{sgn}\left(N_+^{-1} \sum_{o' \in LS_+} \mathbf{a}^T(o') \mathbf{a}(o) - N_-^{-1} \sum_{o' \in LS_-} \mathbf{a}^T(o') \mathbf{a}(o) + b\right)$$

avec $b = \frac{1}{2}(\|\mathbf{c}_-\|^2 - \|\mathbf{c}_+\|^2)$.

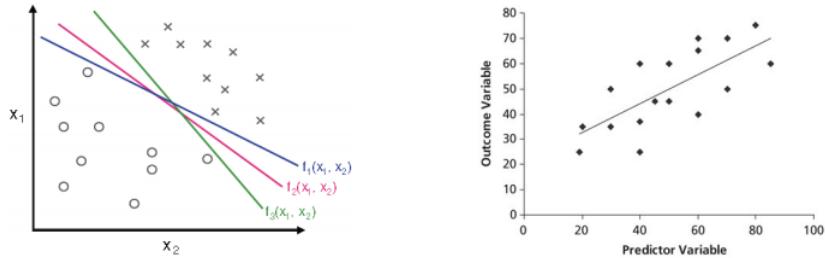
Chapitre 4

Méthodes linéaires

4.1 Introduction

Le but est de trouver un modèle qui est une combinaison linéaire des entrées.

- Pour une régression, $y = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$
- Pour une classification, $y = c_1$ si $w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0$, c_2 sinon.



Plusieurs méthodes existent pour trouver les coefficients w_i :

- Régression : least-square regression, ridge regression, partial least square, support vector regression, LASSO, ...
- Classification : linear discriminant analysis, PLS-discriminant analysis, support vector machines, ...

Avantages et inconvénients :

- + simple ;
- + il existe des variantes rapides et scalables ;
- + cette méthode offre des modèles interprétables, à travers des poids variables (magnitude et signe) ;
- parfois pas aussi précis que d'autres méthodes (non linéaires).

4.2 Ridge regression

Une régression linéaire essaie d'approximer la sortie avec

$$\hat{y}(o) = w_0 + \sum_{i=1}^n w_i a_i(o)$$

Si on pose $a_0(o) = 1 \forall o$, et si on dénote

- $\mathbf{a}'(o) = (a_0(o), a_1(o), \dots, a_n(o))^T$, et

- $\mathbf{w}' = (w_0, w_1, \dots, w_n)^T$,

On a

$$\hat{y}(o) = \mathbf{w}'^T \mathbf{a}'(o)$$

Le carré de l'erreur (Square Error) sur un échantillon i peut s'écrire :

$$SE(o_i, \mathbf{w}') = (y(o_i) - \hat{y}(o_i))^2 = (y(o_i) - \mathbf{w}'^T \mathbf{a}'(o_i))^2$$

On a aussi le carré de l'erreur sur tous les objets de LS (Total Square Error), avec $A' = (\mathbf{a}'_1, \dots, \mathbf{a}'_N)$ et $\mathbf{y} = (y(o_1), y(o_2), \dots, y(o_N))$:

$$TSE(LS, \mathbf{w}') = \sum_{i=1}^N (y(o_i) - \mathbf{w}'^T \mathbf{a}'(o_i))^2 = (\mathbf{y} - A'^T \mathbf{w}')^T (\mathbf{y} - A'^T \mathbf{w}')$$

avec

$$A' = \underbrace{\begin{pmatrix} 1 & 1 & \dots & 1 \\ a_1(o_1) & a_2(o_1) & \dots & a_N(o_1) \\ \vdots & \dots & \dots & \vdots \\ a_1(o_n) & a_2(o_n) & \dots & a_N(o_n) \end{pmatrix}}_N \Bigg\}^{n+1}$$

On cherche \mathbf{w}' qui minimise

$$TSE(LS, \mathbf{w}') = (\mathbf{y} - A'^T \mathbf{w}')^T (\mathbf{y} - A'^T \mathbf{w}')$$

Le gradient est

$$\Delta_{w'} TSE(LS, \mathbf{w}') = -2A'(\mathbf{y} - A'^T \mathbf{w}')$$

Si on résout $\Delta_{w'} TSE(LS, \mathbf{w}') = 0$, on obtient

$$\begin{aligned} 0 &= -2A'\mathbf{y} + 2A'A'^T\mathbf{w}' \\ \Leftrightarrow 2A'\mathbf{y} &= 2A'A'^T\mathbf{w}' \\ \Leftrightarrow \mathbf{w}' &= (A'A'^T)^{-1}A'\mathbf{y} \end{aligned}$$

Il existe cependant plusieurs solutions optimales, ce n'est pas un problème à solution unique. Afin d'avoir une solution unique, on va introduire un terme $\lambda > 0$ de régulation : l'erreur régulée vaut

$$TSE_R(LS, \mathbf{w}') = (\mathbf{y} - A'^T \mathbf{w}')^T (\mathbf{y} - A'^T \mathbf{w}') + \lambda \mathbf{w}'^T \mathbf{w}'$$

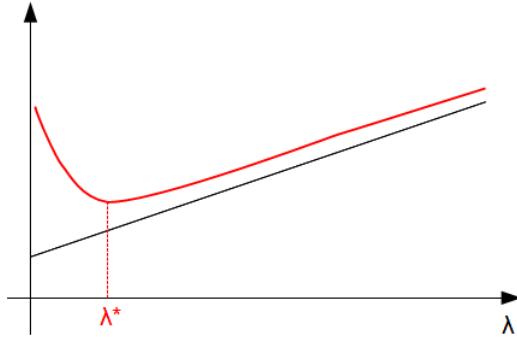
Le gradient devient

$$\Delta_{w'} TSE_R(LS, \lambda, \mathbf{w}') = -2A'(\mathbf{y} - A'^T \mathbf{w}') + 2\lambda \mathbf{w}'$$

La solution optimale devient alors

$$\mathbf{w}^*(\lambda) = (A'A'^T + \lambda I)^{-1}A\mathbf{y}$$

Cette solution est unique $\forall \lambda > 0$. Augmenter λ réduit la variance, car on tient moins compte de A' dans TSE_R . En revanche, on est moins optimal sur l'échantillon, il y a augmentation de l'erreur quadratique moyenne.



Augmenter λ est bénéfique sur un échantillon de test indépendant.

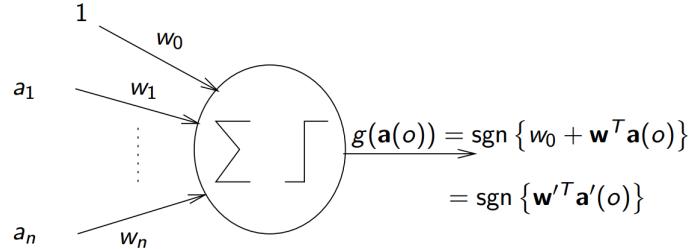
En terme de temps de calcul :

- créer la matrice de covariance est de l'ordre de Nn^2 opérations
- résoudre le système pour trouver \mathbf{w}^* est de l'ordre de n^3 opérations
- $\rightarrow \mathcal{O}(n^3)$

C'est la méthode de régulation qui donne son nom à la *ridge regression*

4.3 Perceptron

L'intuition derrière le perceptron est qu'un cerveau humain peut apprendre, et qu'on pourrait s'en inspirer pour développer un algorithme. Ainsi, un perceptron est la modélisation d'un neurone, qui va ensuite former des couches pour créer des réseaux de neurones.



La sortie est une somme pondérée des entrées. Le seul paramètre à adapter au problème est \mathbf{w}'/\mathbf{w} , qui doit minimiser

$$\sum_i (y_i - w x_i)^2$$

en utilisant une descente de gradient : étant donné un exemple d'entraînement (x, y) ,

$$\begin{aligned} \delta &\leftarrow y - w^T x \\ w_j &\leftarrow w_j + \eta \delta x_j \quad \forall j \end{aligned}$$

Il s'agit d'un algorithme de type *online*, c'est-à-dire qu'il traite les exemples un à un, alors qu'un algorithme de type *batch* traite tous les exemples en une seule fois.

La complexité est régulée par le taux d'apprentissage η et le nombre d'itérations.

4.3.1 Algorithme du perceptron

Pour de la classification binaire, on pose que $c(o) = \pm 1$. On définit η_i le taux d'apprentissage.

- on commence avec un vecteur de poids initial arbitraire, par exemple $\mathbf{w}'_0 = \mathbf{0}$.
- on considère chaque élément de LS dans une séquence cyclique ou aléatoire
- soit l'objet o_i l'objet à l'étape i , $c(o_i)$ est sa classe et $\mathbf{a}(o_i)$ son vecteur d'attributs
- si l'objet o_i est mal classé, on ajuste le vecteur de poids avec la règle suivante :

$$\mathbf{w}'_{i+1} = \mathbf{w}'_i + \eta_i (c(o_i) - g(\mathbf{a}(o_i))) \mathbf{a}'(o_i)$$

4.3.2 Soft threshold units

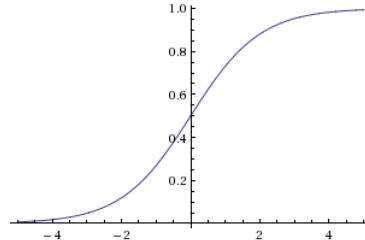
La fonction d'entrée/sortie $g(\mathbf{a})$ est donnée par

$$g(\mathbf{a}(o)) \triangleq f(w_0 + \mathbf{w}^T \mathbf{a}(o)) = f(\mathbf{w}'^T \mathbf{a}'(o))$$

$f(\cdot)$ est une fonction d'activation qui est supposée dérivable. On a généralement pour cette fonction

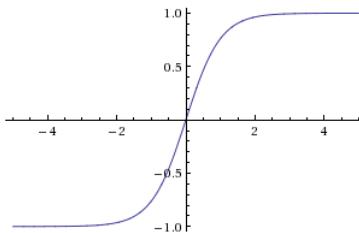
- une sigmoïde :

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

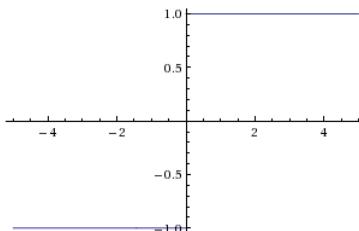


- une tangente hyperbolique :

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



– la fonction sgn



4.3.3 Descente de gradient

Slides 11 et 12/21

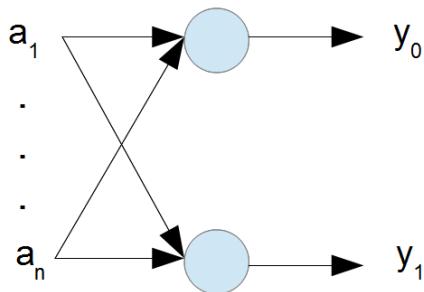
4.3.4 Propriétés

Si LS est linéairement séparable, l'algorithme convergera en un nombre fini d'étape. Sinon, il convergera avec un nombre infini d'étape si $\eta_i \rightarrow 0$.

Résultats théoriques de l'algorithme de descente de gradient : 13/21

4.3.5 Couche de perceptrons

On considère des perceptrons mis côte à côte. Ils ont tous en entrée tous les attributs des objets et fonctionnent de manière indépendante. Par exemple, on pourrait avoir le réseau suivant qui permet d'effectuer une classification parmi quatre classes ; chaque neurone code un bit de la classe.



Cela ne fonctionne toujours que dans le cas de problèmes linéairement séparables, sinon il faut considérer plusieurs couches afin d'obtenir un modèle non linéaire

4.4 Réseaux de neurones

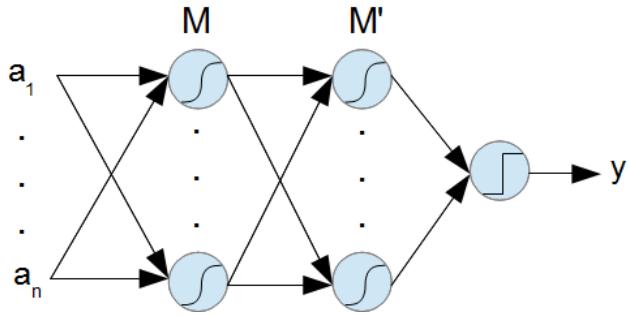
Les réseaux de neurones ont le pouvoir de représentation universel : on peut tout représenter (à un σ près) pour autant qu'on a suffisamment de neurones et de couches.

Un réseau est composé d'au moins une couche d'entrée et d'une couche de sortie, les couches situées entre les deux sont appelées les couches cachées. Chaque neurone est relié à tous les neurones de la couche précédente.

4.4.1 Classification

On considère généralement trois couches :

1. la première peut définir un ensemble d'hyper/demi-plans
2. la seconde peut définir des intersections d'hyper/demi-plans
3. la troisième peut définir des unions d'intersections

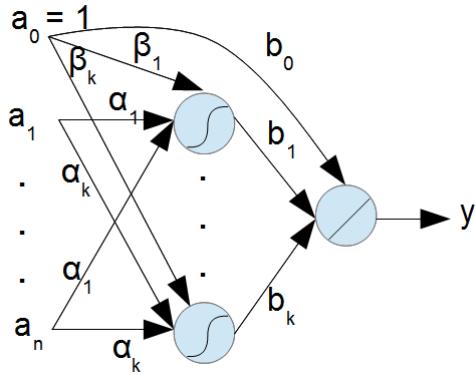


4.4.2 Régression

On considère deux couches :

- la première définit K paramètres d'offset β_i et de scale α_i : les réponses sont $f(\alpha_i x + \beta_i)$ avec $i = 1, \dots, K$
- la seconde couche est une linéarisation :

$$\hat{y}(x) = b_0 + \sum_{i=1}^K b_i f(\alpha_i x + \beta_i)$$



Plus on a de neurones et plus on peut approximer des fonctions continues, avec un K suffisamment grand. La phase d'apprentissage permet d'ajuster les paramètres des fonctions.

4.4.3 Apprentissage d'un réseau multi-couche

On utilise la propagation arrière des dérivations : on commence l'apprentissage à partir de la dernière couche pour revenir à la première. L'idée est de calculer les dérivations de la fonction d'erreur (au sens des moindres carrés) en partant de la couche de sortie jusqu'à la couche d'entrée, ce qui a pour but de réduire la fonction d'erreur en modifiant les poids.

On a l'algorithme suivant pour un réseau de trois couches :

- initialiser les poids du réseau (généralement aléatoirement)
- tant que les exemples ne sont pas correctement classés ou qu'un critère n'est pas satisfait :
 - pour chaque exemple e de l'ensemble d'apprentissage
 - soit o la sortie du réseau de neurones pour l'entrée e (forward pass) et t la "vraie" sortie de e (donnée de l'ensemble d'apprentissage)
 - calculer l'erreur $t - o$
 - calculer les deltas pour tous les poids des synapses qui vont de la couche cachée à la couche de sortie (backward pass)
 - calculer les deltas pour tous les poids des synapses qui vont de la couche d'entrée vers la couche cachée (backward pass continued)
 - mettre à jour les poids du réseau

4.5 Méthodes de régression à base de noyaux

Un noyau $K(., .)$ est une fonction qui exprime la similarité de deux objets à travers une valeur numérique. Pour tout noyau K il existe une fonction ϕ telle que $K(o, o') = \phi(o) \times \phi(o')$.

Lorsqu'on utilise la régression au sens des moindres carrés avec des noyaux, on a un modèle de la forme

$$\hat{y}(x) = w^T \phi(x) + b$$

où on cherche w et b qui minimisent la fonction d'erreur

$$\text{Err}(w, b) = \frac{1}{2} w^T w + \frac{1}{2} \sum_{k=1}^N \underbrace{(y_k - (w^T \phi(x_k) + b))^2}_{e_k}$$

C'est un problème d'optimisation sous contraintes. En passant par le lagrangien on a

$$\begin{aligned} w &= \sum_{j=1}^N \alpha_j \phi(x_j) \\ \alpha_k &= y e_k \\ \sum_{k=1}^N \alpha_k &= 0 \end{aligned}$$

On a donc

$$e_k = y_k - (b + \sum_{j=1}^N \alpha_j K(x_j, x_k))$$

Si on réintroduit e_k dans l'équation 2, on a

$$\hat{y}(x) = b + \sum_{i=1}^N \alpha_i K(x, x_i)$$

avec K une matrice définie par $K_{i,j} = K(x_i, x_j)$. Cela revient à écrire

$$\begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & & & \\ \vdots & & K + \gamma^{-1} I & \\ 1 & & & \end{pmatrix} \begin{pmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix} = \begin{pmatrix} 0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix}$$

L'algorithme d'apprentissage va alors utiliser les valeurs de la matrice K et y_k pour déterminer les α_i et b . Au final, on n'a pas besoin de connaître $\phi(x)$.

Au niveau de la complexité, on a

- N^3 opérations pour l'apprentissage
- N opérations pour la prédiction

Chapitre 5

Biais et variance

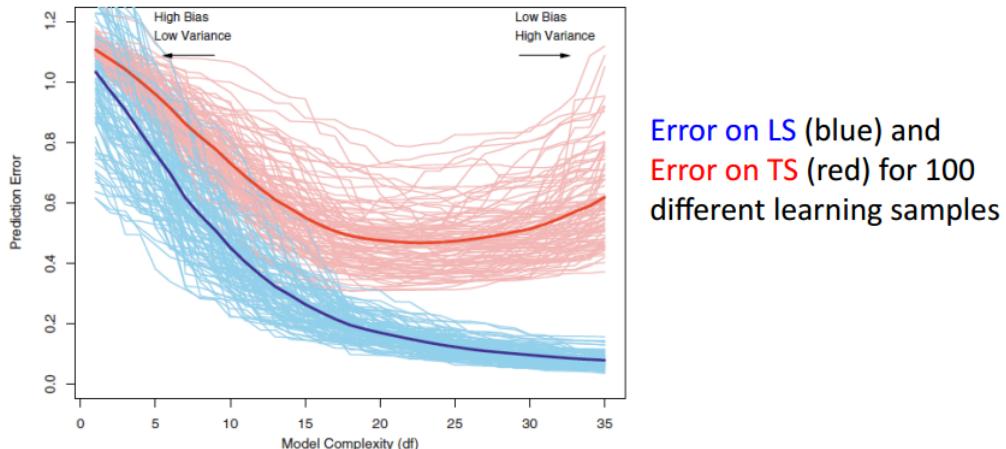
Dans le cadre de l'apprentissage supervisé, on cherche une fonction $f : \mathcal{X} \rightarrow \mathcal{Y}$ qui minimise l'erreur de généralisation, soit l'espérance

$$E_{x,y}\{L(y, f(x))\}$$

La fonction $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ mesure la distance entre ses arguments :

- en classification, $L(y, y') = 1$ si $y \neq y'$ (taux d'erreur)
- en régression, $L(y, y') = (y - y')^2$ (erreur quadratique).

Soit la fonction \hat{f}_{LS} apprise à partir de LS à partir d'un algorithme d'apprentissage. Cette fonction (qui donne la prédiction à un certain point) est une variable aléatoire.



On peut définir l'erreur de généralisation, qui est utile pour caractériser et sélectionner un modèle :

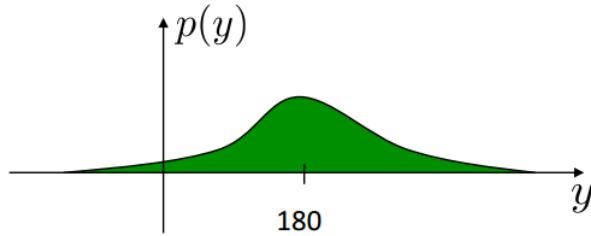
$$Err_{LS} = E_{x,y}\{L(y, \hat{f}_{LS}(x))\}$$

Pour un algorithme d'apprentissage donné, l'erreur de généralisation sur des ensembles LS aléatoires de taille N permet de caractériser un algorithme et est donnée par

$$E_{LS}\{Err_{LS}\} = E_{LS}\{E_{x,y}\{L(y, \hat{f}_{LS}(x))\}\}$$

5.1 Problème de régression sans variable d'entrée

Supposons que l'on cherche à prédire le mieux possible la taille d'un mâle adulte belge. On choisit une mesure de l'erreur, comme l'erreur quadratique, et on cherche une estimation \hat{y} tel que l'espérance $E_y\{(y - \hat{y})^2\}$ sur toute la population belge soit minimale.



L'estimation qui minimise l'erreur peut être calculée en prenant

$$\begin{aligned}
 \frac{\phi}{\phi y'} E_y \{(y - y')^2\} &= 0 \\
 \Leftrightarrow E_y \{-2(y - y')\} &= 0 \\
 \Leftrightarrow E_y \{y\} - E_y \{y'\} &= 0 \\
 \Leftrightarrow y' &= E_y \{y\}
 \end{aligned}$$

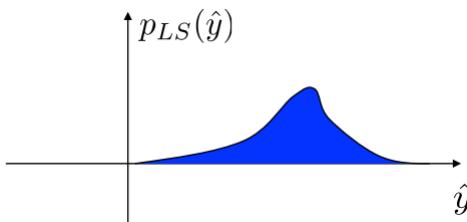
Dans un modèle de Bayes, l'estimation qui minimise l'erreur est $E_y \{y\}$, soit la moyenne sur toute la population. En pratique, ce genre de valeur n'est pas calculable (il faudrait mesurer tous les adultes mâles belges).

Vu que $p(y)$ est inconnu, on va trouver une estimation \hat{y} à partir d'un ensemble $LS = \{y_1, y_2, \dots, y_N\}$, composé d'éléments tirés de manière aléatoire de la population. Autrement dit, on va chercher un algorithme d'apprentissage, par exemple

1. la moyenne : $\hat{y}_1 = \frac{1}{N} \sum_{i=1}^N y_i$
2. en pondérant la valeur qu'on attend, avec λ : $\hat{y}_2 = \frac{\lambda \cdot 180 + \sum_{i=1}^N y_i}{\lambda + N}$, avec $\lambda \in [0, +\infty[$

5.2 Décomposition biais/variance

Vu que LS est tiré aléatoirement de la population, la prédiction \hat{y} est aussi une variable aléatoire. La distribution dépend de l'ensemble de l'apprentissage.



Un bon algorithme d'apprentissage doit être bon sur un seul LS , mais aussi en moyenne sur plusieurs échantillons d'apprentissage. On veut minimiser

$$E = E_{LS} \{E_y \{(y - \hat{y})^2\}\}$$

Si on ajoute et retire $E_y \{y\}$ et si on distribue le carré, on a

$$\begin{aligned}
 E &= E_{LS} \{E_y \{(y - E_y \{y\} + E_y \{y\} - \hat{y})^2\}\} \\
 &= E_{LS} \{E_y \{(y - E_y \{y\})^2\}\} \\
 &\quad + E_{LS} \{E_y \{(E_y \{y\} - \hat{y})^2\}\} \\
 &\quad + E_{LS} \{E_y \{2(y - E_y \{y\})(E_y \{y\} - \hat{y})\}\}
 \end{aligned}$$

Or,

- $E_{LS} \{E_y \{(y - E_y \{y\})^2\}\} = E_y \{(y - E_y \{y\})^2\}$, car aucun terme ne dépend de LS (il n'y a que \hat{y} qui en dépend) ;
- $E_{LS} \{E_y \{(E_y \{y\} - \hat{y})^2\}\} = E_{LS} \{(E_y \{y\} - \hat{y})^2\}$ car $E_y \{y\}$ et \hat{y} sont des constantes dans l'expression, on peut donc les sortir du $E_y \{.\}$ qui les englobe.

$$\begin{aligned}
 E &= E_y \{(y - E_y \{y\})^2\} \\
 &\quad + E_{LS} \{(E_y \{y\} - \hat{y})^2\} \\
 &\quad + E_{LS} \{2E_y \{(y - E_y \{y\})(E_y \{y\} - \hat{y})\}\}
 \end{aligned}$$

Or, $E_y\{y - E_y\{y\}\} = E_y\{y\} - \underbrace{E_y\{E_y\{y\}\}}_{E_y\{y\}}$, le troisième terme s'annule donc, ce qui conduit à

$$E = E_y\{(y - E_y\{y\})^2\} + E_{LS}\{(E_y\{y\} - \hat{y})^2\}$$

$E_y\{(y - E_y\{y\})^2\} = var_y\{y\}$ est la variance des données, c'est l'erreur résiduelle ; il s'agit de la plus petite erreur que l'on peut atteindre. Il s'agit d'une donnée du problème, on ne peut pas agir dessus.

$E_{LS}\{(E_y\{y\} - \hat{y})^2\}$ est la moyenne sur tous les ensembles d'apprentissage de l'erreur du modèle de Bayes.

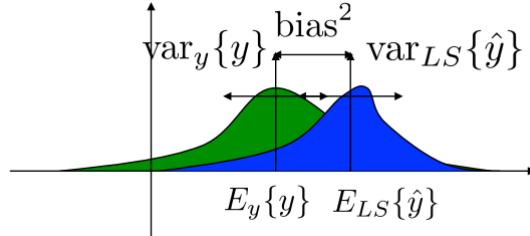
Si on suit le même raisonnement

$$\begin{aligned} E_{LS}\{(E_y\{y\} - \hat{y})^2\} &= E_{LS}\{(E_y\{y\} - E_{LS}\{\hat{y}\} + E_{LS}\{\hat{y}\} - \hat{y})^2\} \\ &= E_{LS}\{(E_y\{y\} - E_{LS}\{\hat{y}\})^2\} \\ &\quad + E_{LS}\{(E_{LS}\{\hat{y}\} - \hat{y})^2\} \\ &\quad + E_{LS}\{2(E_y\{y\} - E_{LS}\{\hat{y}\})(E_{LS}\{\hat{y}\} - \hat{y})\} \\ &= (E_y\{y\} - E_{LS}\{\hat{y}\})^2 \\ &\quad + E_{LS}\{(\hat{y} - E_{LS}\{\hat{y}\})^2\} \\ &\quad + 2(E_y\{y\} - E_{LS}\{\hat{y}\})(E_{LS}\{\hat{y}\} - \hat{y}) \\ &= (E_y\{y\} - E_{LS}\{\hat{y}\})^2 + E_{LS}\{(\hat{y} - E_{LS}\{\hat{y}\})^2\} \end{aligned}$$

$E_{LS}\{\hat{y}\}$ est la moyenne du modèle sur tout l'échantillon LS . $(E_y\{y\} - E_{LS}\{\hat{y}\})^2$ est le biais², l'erreur entre le modèle de Bayes et le modèle moyen.

$var_{LS}\{\hat{y}\}$ est l'estimation de la variance, qui est la conséquence de sur-apprentissage. On a que

$$\begin{aligned} E &= var_y\{y\} + \text{biais}^2 + E_{LS}\{(\hat{y} - E_{LS}\{\hat{y}\})^2\} \\ &= var_y\{y\} + \text{biais}^2 + var_{LS}\{\hat{y}\} \end{aligned}$$



Pour les exemples :

$$-\hat{y}_1 = \frac{1}{N} \sum_{i=1}^N y_i$$

$$E_{LS}\left\{\frac{1}{N} \sum_{i=1}^N y_i\right\} = \frac{1}{N} \sum_{i=1}^N E_{LS}\{y_i\} = \frac{1}{N} \sum_{i=1}^N E_y\{y\} = E_y\{y\}$$

$E_{LS}\{y_i\}$ est la taille moyenne d'une même personne pour plusieurs LS , tandis que $E_y\{y\}$ est la taille moyenne de la population : ce sont les mêmes choses, car la distribution est iid, le choix ne fait pas varier la taille.

$$var_{LS}\{\hat{y}\} = var_{LS}\left\{\frac{1}{N} \sum_{i=1}^N y_i\right\} = \frac{1}{N^2} var_{LS}\left\{\sum_{i=1}^N y_i\right\}$$

Par une propriété.

$$= \frac{1}{N^2} \sum_{i=1}^N var_{LS}\{y_i\}$$

car les y_i sont indépendants.

$$= \frac{1}{N^2} N var_y\{y\}$$

car la distribution est iid.

Les caractéristiques de cet estimateur sont que

1. si $N \rightarrow +\infty$, alors $E \rightarrow 0$.
 2. le biais est nul, c'est le meilleur estimateur.
- $\lambda : \hat{y}_2 = \frac{\lambda \cdot 180 + \sum_{i=1}^N y_i}{\lambda + N}$, avec $\lambda \in [0, +\infty[$

$$\text{biais}^2 = \frac{\lambda}{\lambda + N} (E_y\{y\} - 180)^2$$

$$var_{LS}\{\hat{y}_2\} = \frac{N}{(\lambda + N)^2} var_y\{y\}$$

Caractéristiques :

- le biais est petit si la taille moyenne d'un homme belge est proche de 180
 - plus l'échantillon est grand, plus le biais est petit
 - si λ est élevé, la variance va diminuer (car la prédiction devient constante), mais le biais va augmenter.
- \hat{y}_1 et \hat{y}_2 sont des estimateurs consistants car lorsque N augmente (donc lorsqu'on a plus d'échantillons), E diminue, ce qui n'est pas toujours le cas. \hat{y}_2 combat le sur-apprentissage, on dit que λ permet la régulation.

5.2.1 Approche bayesienne

Supposons que

- la taille moyenne est proche de 180 cm, suivant une loi normale :

$$P(\bar{y}) = A \exp\left(-\frac{(\bar{y} - 180)^2}{2\sigma_{\bar{y}}^2}\right)$$

- la taille d'une personne est gaussienne par rapport à la moyenne :

$$P(y_i|\bar{y}) = B \exp\left(-\frac{(y_i - \bar{y})^2}{2\sigma_y^2}\right)$$

Quelle est la valeur la plus probable \bar{y} en connaissant LS ?

$$\hat{y} = \arg \max_{\bar{y}} P(\bar{y}|LS)$$

Voir slide 20.

5.2.2 Problème de régression (complet)

On considère qu'on cherche une fonction $\hat{y}(x)$ à plusieurs entrées, on a une moyenne sur tout l'espace d'entrée. L'erreur devient

$$\begin{aligned} E &= E_{LS}\{\mathbb{E}_{\underline{x}, y}(y - \hat{y}(\underline{x}))^2\} = \mathbb{E}_{\underline{x}} E_{LS}\{\mathbb{E}_{y|\underline{x}}\{(y - \hat{y}(\underline{x}))^2\}\} \\ &= \mathbb{E}_{\underline{x}} var_{y|\underline{x}}\{y\} + \mathbb{E}_{\underline{x}} \text{biais}^2(\underline{x}) + \mathbb{E}_{\underline{x}} var_{LS}\{\hat{y}(\underline{x})\} \end{aligned}$$

On a finalement l'erreur suivante :

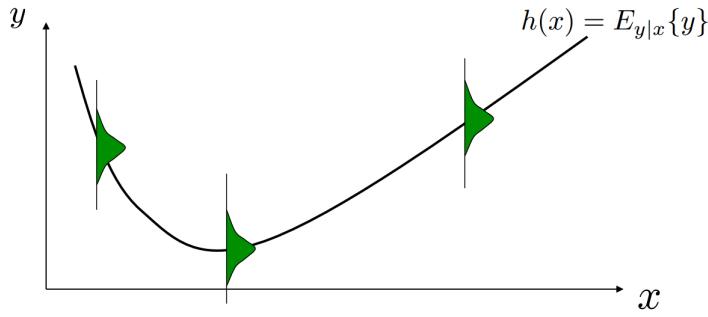
$$E_{LS}\{\mathbb{E}_{y|\underline{x}}\{(y - \hat{y}(\underline{x}))^2\}\} = \text{noise}(\underline{x}) + \text{biais}^2(\underline{x}) + \text{variance}(\underline{x})$$

avec

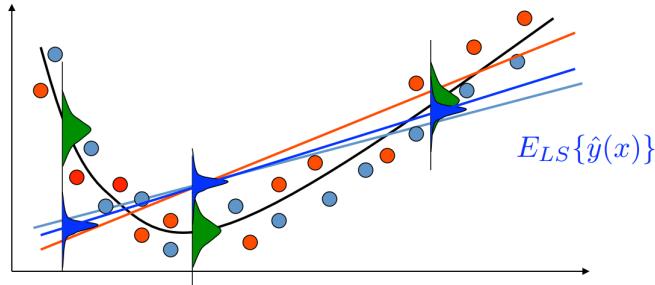
- $\text{noise}(\underline{x}) = \mathbb{E}_{y|\underline{x}}\{(y - h_B(\underline{x}))^2\}$, qui quantifie de combien y varie de $h_B(\underline{x}) = \mathbb{E}_{y|\underline{x}}\{y\}$, le modèle de Bayes
- $\text{biais}^2(\underline{x}) = (h_B(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2$, qui mesure l'erreur entre le modèle de Bayes et la moyenne du modèle
- $\text{variance}(\underline{x}) = E_{LS}\{(\hat{y} - E_{LS}\{\hat{y}(\underline{x})\})^2\}$, qui quantifie de combien $\hat{y}(\underline{x})$ varie d'un échantillon d'apprentissage à un autre.

5.2.3 Illustration

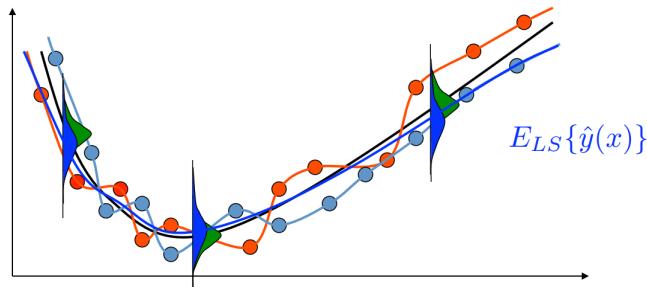
Soit une fonction $y = h(x) + \epsilon$, $\epsilon \approx N(0, 1)$.



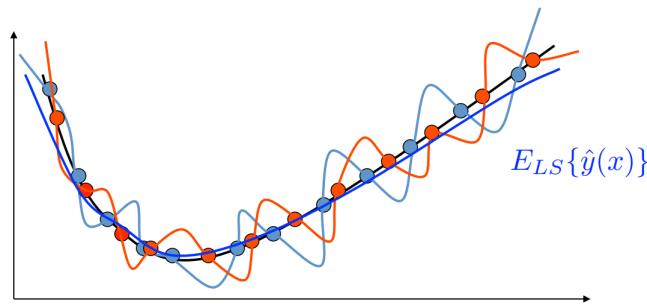
Si on utilise une méthode avec trop de biais et peu de variance, on a du sous-apprentissage. Le biais est la différence au carré entre les courbes bleu et noire (intégrée).



Si on utilise une méthode avec peu de biais et beaucoup de variance, on a du sur-apprentissage. Le décalage est faible, mais la variance est haute. La variance est proche du bruit.



A noter que ne pas avoir de bruit ne signifie pas qu'il n'y a pas de variance, mais qu'il y en a moins.



5.2.4 Lien entre biais/variance et sous/sur-apprentissage

Un modèle trop simple ne permet pas de capturer toute la complexité des données : on a un biais élevé.

Un modèle trop complexe conduit à du sur-apprentissage : on a une variance élevée.

5.3 Problèmes de classification

L'erreur moyenne de mauvaise classification est

$$E = E_{LS}\{E_{\underline{x},y}\{1(y \neq \hat{y}(\underline{x}))\}\}$$

Le meilleur modèle possible est le modèle de Bayes :

$$h_B(\underline{x}) = \arg \max_c P(y = c | \underline{x})$$

Le modèle moyen est

$$\arg \max_c P(\hat{y}(\underline{x}) = c | \underline{x})$$

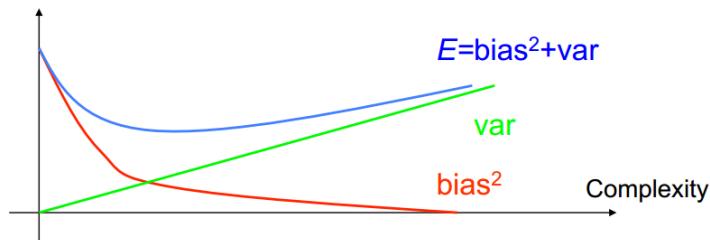
Il n'y a pas de décomposition de l'erreur de mauvaise classification moyenne en un biais et une variance, mais on observe les mêmes phénomènes.

Le biais est à peu près une composante de l'erreur qui est systématique et indépendante de l'échantillon d'apprentissage. La variance est l'erreur due à la variabilité du modèle par rapport à l'aspect aléatoire de l'échantillon d'apprentissage.

5.4 Paramètres influençant le biais et la variance

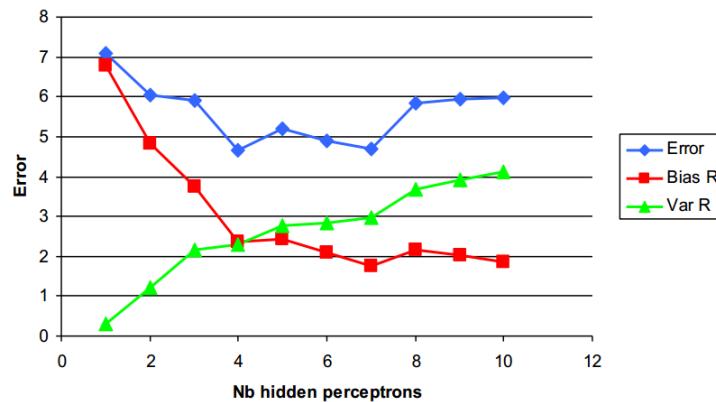
5.4.1 Complexité du modèle

Pour le problème considéré, on a que la variance du bruit vaut 1 et la moyenne est nulle : l'erreur résiduelle vaut donc 1.



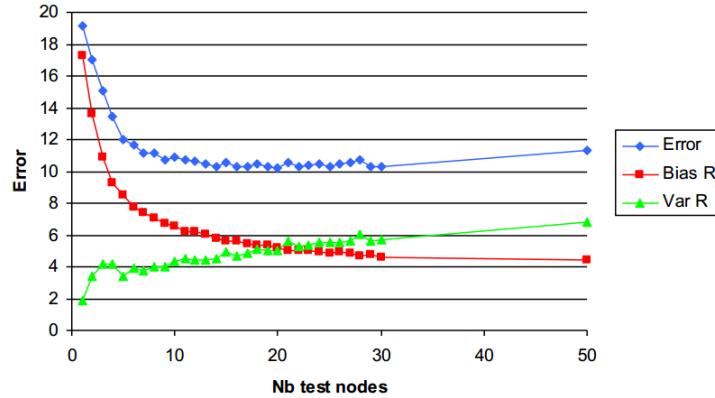
Généralement, le biais est une fonction décroissante de la complexité, tandis que la variance croît.

Réseaux de neurones



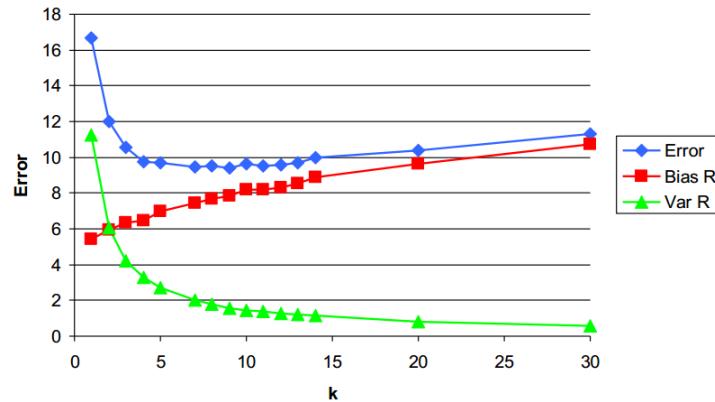
Plus on augmente le nombre de neurones dans la couche cachée et plus on obtient une fonction complexe. Du coup, on tient forcément plus compte de bruit ou d'outliers, ce qui entraîne une augmentation de la variance dans les prédictions. Le biais va diminuer pour la même raison ; on approxime de mieux en mieux la fonction.

Arbres de régression



Le raisonnement est similaire à celui des réseaux de neurones : plus l'arbre est profond et plus on tient compte du bruit et de cas isolés dans les données. De ce fait, on aura une erreur de resubstitution (donc le biais) moindre, mais la variance dans les prédictions sera plus grande.

k -NN

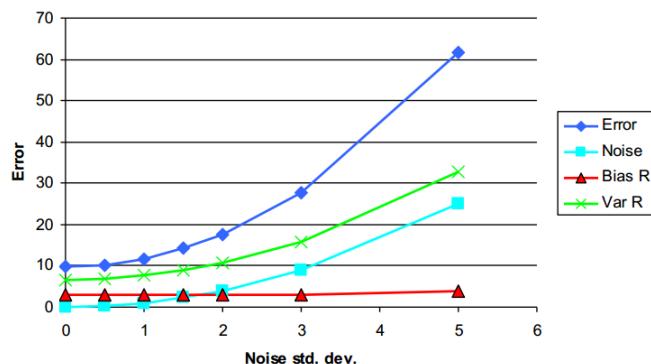


Plus k est grand et plus on considère des voisins éloignés. Il est logique que la variance diminue, car on considère une fraction de plus en plus grande des données dans le calcul de la prédiction, du coup les prédictions varient de moins en moins (si on considérait tous les points, la prédiction serait la même tout le temps). En revanche, vu qu'on considère des voisins qui ont peu de chance d'être similaires à l'entrée, on aura un plus grand biais.

5.4.2 Bruit

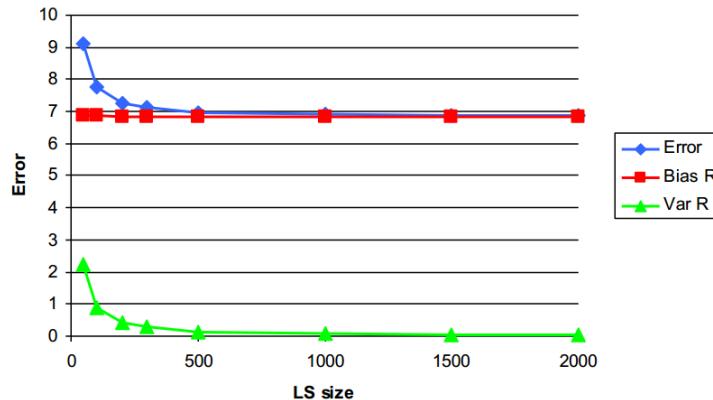
A une complexité de modèle fixée, le biais augmente avec la complexité du modèle de Bayes. Cependant, les effets sur la variance sont difficiles à prédire.

A cause du bruit, la variance augmente, mais globalement le biais n'est pas affecté. Par exemple, avec un arbre de régression (complet).

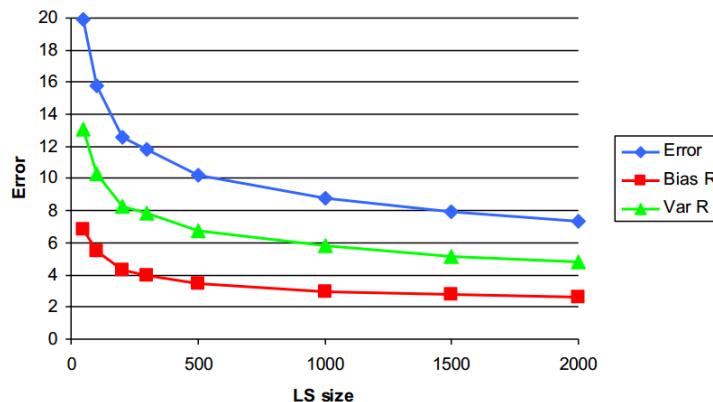


5.4.3 Taille de l'échantillon d'apprentissage

Pour les modèles dont la complexité ne dépend pas de la taille de LS, le biais reste constant et la variance diminue avec la taille de l'échantillon. Par exemple, avec une régression linéaire.



Lorsque la complexité du modèle est dépendante de la taille de l'échantillon d'apprentissage (par exemple les arbres), le biais et la variance décroissent avec la taille de l'échantillon d'apprentissage. Par exemple, avec un arbre de régression.



5.4.4 Algorithmes d'apprentissage

Method	Err ²	Bias ² +Noise	Variance
Linear regr.	7.0	6.8	0.2
k-NN (k=1)	15.4	5	10.4
k-NN (k=10)	8.5	7.2	1.3
MLP (10)	2.0	1.2	0.8
MLP (10 – 10)	4.6	1.4	3.2
Regr. Tree	10.2	3.5	6.7

- la régression linéaire possède peu de paramètres, donc une petite variance. Par contre, si la classification est non linéaire, on a une variance élevée
- k-NN : un petit k implique une variance élevée et un biais modéré. Un grand k implique une petite variance mais un biais plus grand
- réseau de neurones : le biais est petit, mais la variance augmente avec la complexité du modèle
- arbre de régression : le biais est petit, mais la variance est grande.

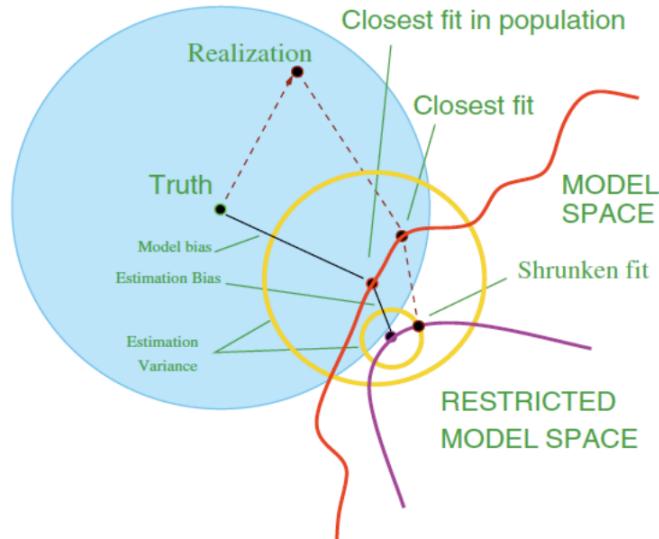
5.5 Techniques de réduction du biais et de la variance

Afin de réduire le biais et la variance, on peut jouer sur les paramètres de l'algorithme d'apprentissage, cependant cela ne suffit pas toujours. On peut alors passer aux méthodes d'ensemble, qui offrent un trade-off biais/variance différent, mais qui font perdre des caractéristiques de la méthode initiale.

5.5.1 Réduction de la variance

L'idée générale est de réduire la capacité de l'algorithme d'apprendre LS. Cela se fait par

- du pruning, afin de réduire explicitement la complexité de l'algorithme
- un arrêt précoce, afin de réduire la recherche
- une régularisation, qui réduit l'espace d'hypothèses. On utilise par exemple le weight decay avec les réseaux de neurones, qui consiste à pénaliser les grandes valeurs pour les poids.



La réalisation est la vraie valeur auquel s'ajoute du bruit. Le biais du modèle n'est pas équivalent au biais de l'algorithme d'apprentissage.

Exemple de résultats :

Method	E	Bias	Variance
Full regr. Tree (250)	10.2	3.5	6.7
Pr. regr. Tree (45)	9.1	4.3	4.8
Full learned MLP	4.6	1.4	3.2
Early stopped MLP	3.8	1.5	2.3

Comme attendu, la variance diminue mais le biais augmente.

A noter qu'une bonne estimation du biais est l'erreur de resubstitution (sur LS).

Chapitre 6

Évaluation de modèles

On aimerait estimer les performances d'un modèle ayant appris sur un ensemble de données (de taille N). Le but est de

- sélectionner un ou plusieurs modèles (ex : déterminer la bonne complexité ou choisir entre différents algorithmes d'apprentissage).
- évaluer les modèles, afin d'estimer les performances sur des nouvelles données.

6.1 Erreurs

6.1.1 Erreur de resubstitution

L'erreur de re-substitution (LS error) est l'erreur obtenue en appliquant le modèle à l'échantillon d'apprentissage. Plus le modèle est complexe et plus cette erreur sera proche de 0.

Cette erreur est un mauvais indicateur des performances d'un modèle car il est beaucoup trop optimiste.

6.1.2 Erreur de généralisation

L'erreur de généralisation est l'erreur obtenue sur la prédiction de nouvelles données. Si \hat{f}_{LS} est la fonction apprise sur un échantillon LS, l'erreur de généralisation est décrite par

$$\mathbb{E}_{\underline{x},y}\{L(y, \hat{f}_{\text{LS}}(\underline{x}))\}$$

avec $L(.,.)$ une fonction de perte qui mesure la différence entre les arguments. L'erreur de généralisation est utilisée pour l'évaluation et la sélection de modèle.

Cette erreur est à différencier de l'erreur de généralisation attendue (expected generalization error) sur un LS aléatoire de taille N , décrite par

$$E_{\text{LS}}\{\text{Err}_{\text{LS}}\} = E_{\text{LS}}\{\mathbb{E}_{\underline{x},y}L(y, \hat{f}_{\text{LS}}(\underline{x}))\}$$

Cette dernière à pour but de décrire le comportement d'un algorithme.

6.2 Méthodes d'évaluation

La validation croisée a pour but de prédire les performances d'un modèle avec des données d'apprentissage lorsque des nouvelles données explicites ne sont pas disponibles.

6.2.1 Méthode test set

On suppose qu'on dispose de beaucoup de données, que N est grand. On divise la base de données en deux parties, une qui servira d'ensemble d'apprentissage et l'autre d'ensemble de test (par ex 70%, 30%). La méthode est la suivante :

- on apprend le modèle sur LS
- on le test sur TS
- l'estimation qui en résulte est une estimation de l'erreur d'un modèle qui aurait appris sur toute la base de données.

Avantages et inconvénients :

- + très simple à mettre en place ;
- + efficace ;

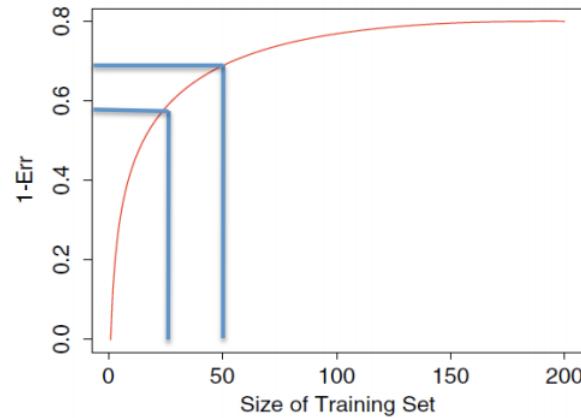
- les données d'apprentissage sont moindres ;
- instable lorsque la base de données est petite.



6.2.2 K-fold

La méthode du test-set n'est pas fiable sur une petite base de données car elle est basée sur un petit échantillon d'une base de données déjà réduite. De plus, on utilise l'estimation du modèle construit comme une estimation pour toute la base de données, or lorsque la base de données est très petite, apprendre le modèle sur toute la base ou sur une partie change fortement les résultats.

On peut tracer la courbe d'apprentissage où on confronte les performances à la taille de l'échantillon d'apprentissage. On voit qu'il faut que le LS soit d'une taille minimale si on veut avoir des résultats pertinents.



On utilise pour cela la validation k-fold : on divise aléatoirement la base de données en k sous-ensembles (typiquement $k = 10$).



La méthode est la suivante :

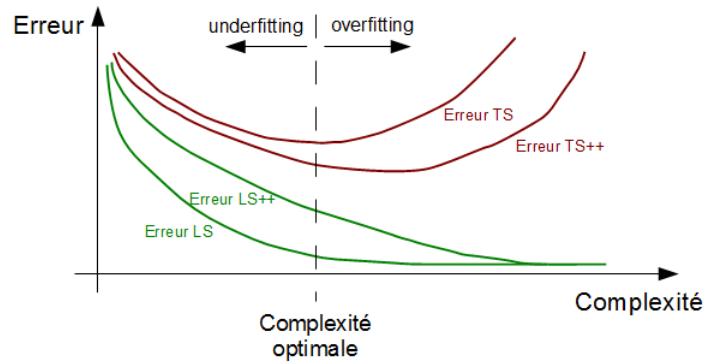
- pour chaque sous-ensemble
 - apprendre le modèle sur les objets qui ne sont pas dans le sous-ensemble
 - calculer les prédictions du modèle sur les points du sous-ensemble
 - reporter l'erreur moyenne sur ces prédictions

Lorsque $k = N$, la méthode s'appelle une validation croisée *leave-one-out*.

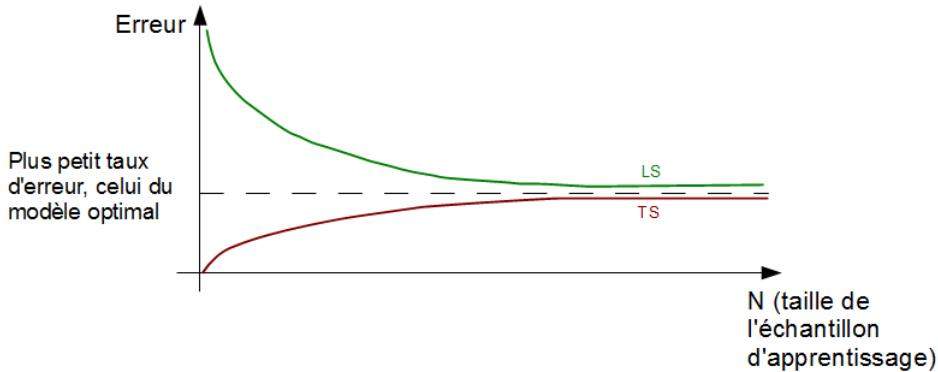
Le choix de k est très important et conduit à différents avantages et inconvénients :

- si $k = N$ (*leave-one out*) :
 - + non-biaisé : enlever un objet ne change pas trop la taille de l'échantillon d'apprentissage
 - grande variance : on dépend énormément de la base de données
 - lent : il faut entraîner N modèles
- si $k = 5, 10$:
 - + petite variance et rapide : on n'a que 5 – 10 modèles sur peu de données
 - potentiellement biaisé (voir courbe d'apprentissage)

6.2.3 Impact de la complexité d'un modèle avec une validation croisée



Il vaut mieux utiliser une petite complexité si on ne dispose pas de beaucoup d'échantillons.
Avec une complexité fixe, on obtient le comportement suivant.



Le contrôle de la complexité s'appelle la régulation ou le lissage (smoothing). Il peut être contrôlé de plusieurs façons :

- en variant la taille de l'espace d'hypothèse, autrement dit le nombre de modèles candidats, la valeur des paramètres, etc ;
- avec un critère de performance : on oppose les performances de l'ensemble d'apprentissage et la valeur des paramètres, autrement dit on minimise

$$\text{Err}(LS) + \lambda C(\text{model})$$

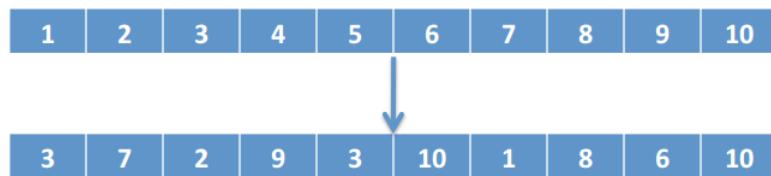
- avec des algorithmes d'optimisation : le nombre d'itération, la nature du problème d'optimisation, etc.

Le choix d'un algorithme se fait en comparant leur taux d'erreur avec une méthode de type cross-validation sur des sous-échantillons. Ensuite, l'algorithme avec le plus petit taux est utilisé comme modèle prédictif sur toutes les données.

L'utilisation intensive de la méthode CV peut entraîner du sur-apprentissage. En effet, plus on compare des modèles complexes, plus on a une chance d'en trouver un qui convient pour les données. La solution pour éviter cela est de réservé un ensemble de test additionnel (ou de les générer), et de l'utiliser pour tester les performances du modèle final.

6.2.4 Bootstrap

Un échantillon bootstrap est un échantillon avec un remplacement : des objets n'apparaissent pas et d'autres apparaissent plusieurs fois.



On a alors que

$$P(o_i \in \text{bootstrap}) = 1 - (1 - \frac{1}{N})^N \approx 1 - \frac{1}{e} = 0.632$$

L'idée est d'utiliser les 30% de données qu'il reste comme TS. On peut alors estimer l'erreur de bootstrap :

- pour $i = 1$ jusqu'à B :
 - prendre un échantillon de bootstrap B_i de la base de données
 - apprendre un modèle f_i sur cet échantillon
 - pour chaque objet, calculer l'erreur de tous les modèles qui ont été construits sans lui (environ 30%)
 - moyenne sur tous les objets
- Des améliorations existent :
- .632 bootstrap : correction pour la courbe d'apprentissage
 - .632+ bootstrap : correction pour le sur-apprentissage

6.2.5 Erreurs de test conditionnelles et erreurs de test attendues

Pour un modèle \hat{f}_{LS} donné, on a l'erreur de test conditionnelle :

$$\text{Err}_{LS} = E_{x,y} L(y, \hat{f}_{LS}(x))$$

On a l'erreur attendue :

$$E_{LS}\{\text{Err}_{LS}\} = E_{LS}\{E_{x,y} L(y, \hat{f}_{LS}(x))\}$$

Seule la méthode test set estime la première erreur, la validation croisée permet quant à elle d'estimer la seconde car on fait de l'apprentissage sur des LS différents.

6.3 Méthodes de sélection

Le but, pour une base de données de N objets, est de déterminer le meilleur modèle possible et d'estimer l'erreur des prédictions. La méthodologie dépend encore une fois de la taille de la base.

Le choix d'une mesure de l'erreur ou de la qualité dépend fortement de l'application. On peut également définir des autres critères pour évaluer un modèle.

6.3.1 Méthode test set

Si la base de données est grande, on divise aléatoirement l'ensemble d'apprentissage en trois parties : un ensemble d'apprentissage LS, un ensemble de validation VS et un ensemble de test TS (par exemple 50%, 25% et 25%).



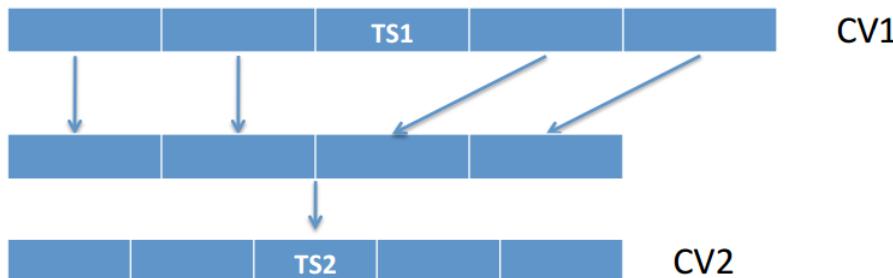
La méthode est la suivante :

- apprendre les modèles à comparer sur LS
- sélectionner le meilleur en basant les performances sur VS
- le ré-entraîner sur LS + VS
- le tester sur TS, afin d'avoir une estimation des performances
- le ré-entraîner sur LS + VS + TS, afin d'avoir le modèle final

VS permet de sélectionner le modèle. Une fois que c'est fait, on réapprend sur LS et VS et on teste sur TS. Le choix de la meilleure méthode (apprise sur LS et testée sur VS) n'entraîne pas de sur-apprentissage, mais il y a tout de même un risque s'il y a trop de méthodes à tester (on en aurait une qui donne des bons résultats par coup de chance), ce qui pourrait donner une grosse erreur sur TS. La solution est d'ajouter une nouvelle boucle de validation croisée ; tout choix d'échantillon peut créer un biais.

6.3.2 K-fold

On utilise deux étapes de validation croisée en k-fold.



CV1 est utilisé pour évaluer le modèle final, tandis que CV2 est utilisé pour la sélection du modèle. On peut également combiner la méthode test set et la validation croisée.



Les deux étapes sont nécessaires, car plus on compare des modèles et plus la probabilité de tomber par chance sur celui qui donne des bons résultats augmente. Les erreurs sur VS ou sur CV2 sont alors en général trop optimistes.

Illustration :

- Dataset: N=50, 1000 inputs variables, all unrelated to the class (values are randomly drawn from $N(0,1)$)
=> any model should have a 50% generalization error rate
- Comparison of 1000 learning algorithms: i th algorithm learns a decision tree on the i th feature only
- 10-fold CV error of the best model: 16%
- Its error on a test sample of 5000 cases: 48%

6.3.3 Méthodes analytiques

On cherche le modèle qui minimise un critère de la forme

$$\text{Err(LS)} + G(\text{complexité})$$

où G est une fonction monotone croissante. Le critère est dérivé de preuves théoriques.

L'avantage est qu'il n'y a pas de re-entraînement, par contre on ne peut l'utiliser que pour la sélection de modèle, et on pourrait manquer le vrai optimum dans le cas d'échantillons finis.

6.4 Biais de sélection

En général, n'importe quel choix fait en utilisant la sortie doit être dans une boucle de validation croisée, car il peut potentiellement amener à du sur-apprentissage. En effet, un gain d'apprentissage sur un LS peut ne pas se répercuter sur des nouvelles données à prédire.

- Another example: on the same dataset:
 - Select the 10 attributes that are the most correlated with the output on the LS
 - Estimate the error rate of a tree built with these 10 attributes using 10-fold CV on the same LS => 20%
 - Error of this model on a sample of 5000 cases => 51% (this problem is called the selection bias)

6.5 Mesure de performance

6.5.1 Classification binaire

Les résultats peuvent être résumés dans un tableau de contingence/matrice de confusion.

		Predicted class		Total
Actual class	p	p	n	
		True Positive	False Negative	
p				P
n		False Positive	True Negative	N

On définit alors

$$\text{Taux d'erreur (error rate)} = \frac{FP + FN}{N + P}$$

$$\text{Précision (accuracy)} = \frac{TP + TN}{N + P} = 1 - \text{error rate}$$

Le taux d'erreur est cependant limité : on n'a pas d'informations sur la distribution des erreurs sur les classes, et il est sensible aux changements dans la distribution des classes dans l'échantillon de test.

Model 1			Model 2			Model 3			
Actual class	Predicted class		Actual class	Predicted class		Actual class	Predicted class		
	p	n	Total	p	n	Total	p	n	Total
p	0	10	10	10	0	10	0	50	50
n	0	90	90	10	80	90	0	50	50

Error rate=10% Error rate=10% Error rate=50%

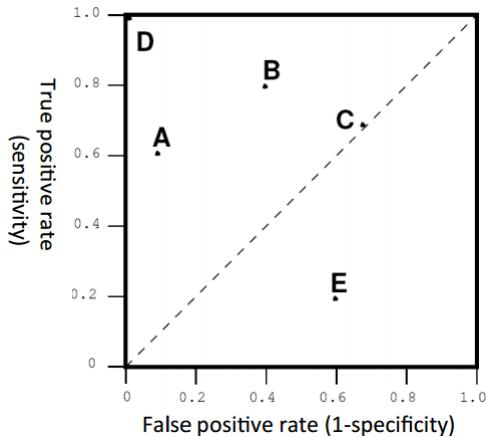
Dans le cadre de diagnostics médicaux, on utilise des mesures plus appropriées :

$$\text{Sensibilité (sensitivity/recall)} = \frac{TP}{P}$$

$$\text{Spécificité (specificity)} = \frac{TN}{TN + FP} = 1 - \frac{FP}{N}$$

La sensibilité permet de détecter le maximum de positif (plus elle est grande et plus on est sûr de détecter les cas positifs), tandis que la spécificité détecte le maximum de négatif. L'avantage de ces mesures est qu'elles ne dépendent pas de la proportion d'objets positifs ou négatifs.

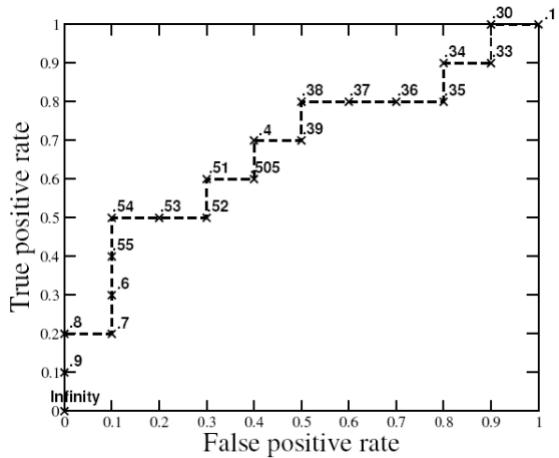
Ces mesures peuvent être portées sur une courbe ROC (Receiver operating characteristic). Si la sortie de l'algorithme d'apprentissage est un nombre (par exemple, la probabilité d'appartenir à une classe), on peut utiliser un seuil afin de régler la sensibilité et la spécificité.



Le meilleur algorithme est le D. Si un modèle retourne toujours la classe positive, il se trouvera dans le coin supérieur droit. S'il retourne toujours la classe négative, il sera dans le coin inférieur gauche. S'il renvoie une valeur au hasard, il sera situé au centre ; la diagonale représente les choix aléatoires biaisés.

Si on se trouve dans la diagonale inférieure, on peut inverser les réponses ($+ \Rightarrow -, - \Rightarrow +$) pour revenir dans la diagonale supérieure.

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

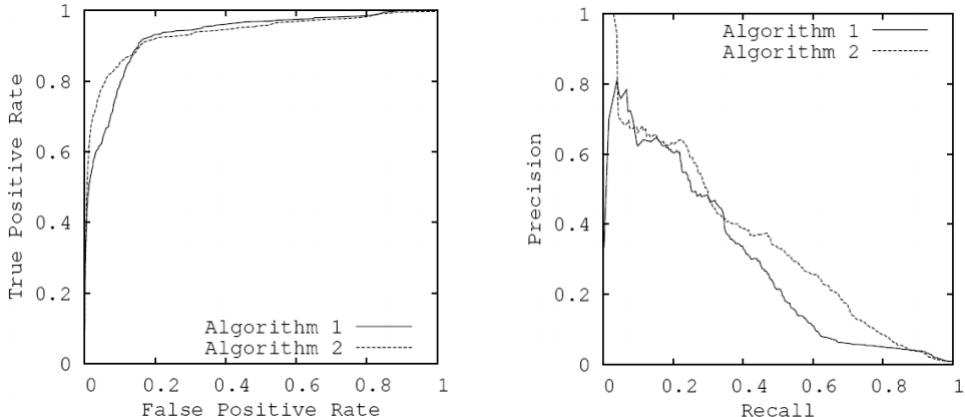


On peut résumer une courbe ROC avec un nombre : l'AUC, la surface en dessous de la courbe. On peut l'interpréter comme la probabilité que deux objets choisis aléatoirement dans l'échantillon sont correctement ordonnés par le modèle, c'est-à-dire que le positif a un plus haut score que le négatif.

On utilise d'autre mesures :

- la précision, la proportion de bonnes prédictions parmi les prédictions positives
- le *recall*, la proportion de positifs qui sont détectés
- la *f-measure*

$$\begin{aligned} \text{Précision} &= \frac{TP}{TP + FP} \\ \text{Recall} &= \frac{TP}{TP + FN} \\ \text{F-mesure} &= \frac{2 * \text{précision} * \text{recall}}{\text{précision} + \text{recall}} \end{aligned}$$



Le meilleur algorithme aura une courbe ROC de la forme \lceil , tandis que la courbe de recall aura la forme \lfloor .

6.5.2 Régression

A partir du moment où on a plus d'une classe, on utilise un taux d'erreur plutôt qu'une courbe ROC.

Erreur quadratique

$$\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Le carré permet de pénaliser les très mauvaises prédictions.

Erreur absolue moyenne

$$\frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

Corrélation de Pearson

$$\frac{\sum_i (y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{(N-1)s_y s_{\hat{y}}}$$

Corrélation de rang de Spearman

$$1 - \frac{6 \sum_i d_i^2}{N(N^2 - 1)}$$

avec d_i la différence de rang de y_i et \hat{y}_i , le rang étant l'ordre obtenu après un tri des valeurs.

6.5.3 Mesures de performances pour l'entraînement

Les mesures de performances pour l'entraînement peuvent être différentes des mesures de performances de test. Il y a plusieurs raisons à cela :

- algorithmiquement, une mesure dérivable est soumise à une optimisation de gradient (par exemple le taux d'erreur et l'erreur absolue moyenne ne sont pas dérivables, l'AUC n'est pas décomposable)
- le sur-apprentissage : pour l'entraînement, la perte incorpore souvent un terme de pénalité pour la complexité du modèle (ce qui est inutile au moment du test). De plus, certaines mesures sont moins promptes au sur-apprentissage (par exemple la marge).

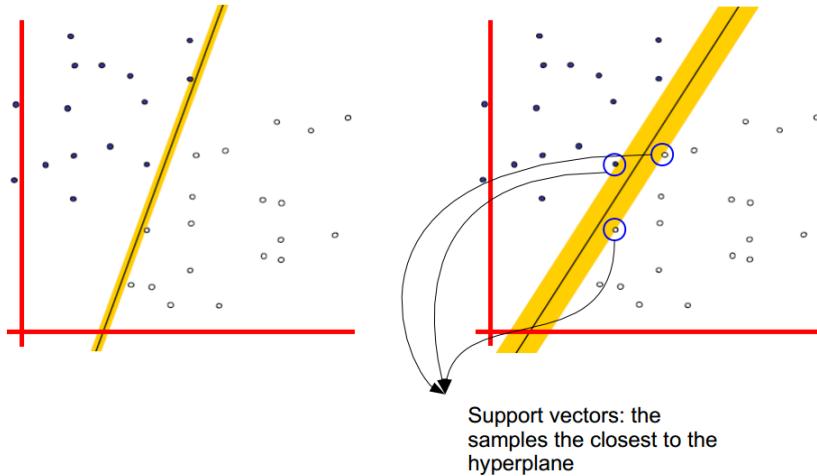
Chapitre 7

Machines à support vectoriel

Cette méthode se base sur deux idées :

1. la mise en place d'un classificateur à large marge, et
2. le "noyautage" de l'espace d'entrée.

Il faut trouver un classificateur linéaire. L'idée va être de trouver celui qui maximise la marge, c'est-à-dire la largeur du bord qui peut être étendue jusqu'à toucher une donnée.



Support vectors: the samples the closest to the hyperplane

7.1 Machines à support vectoriel linéaire

Soit un LS = $\{(x_k, y_k)\}_{k=1}^N$, avec $y_k \in [-1, 1]$ et $x_k \in \mathbb{R}^n$. On cherche un classificateur de la forme

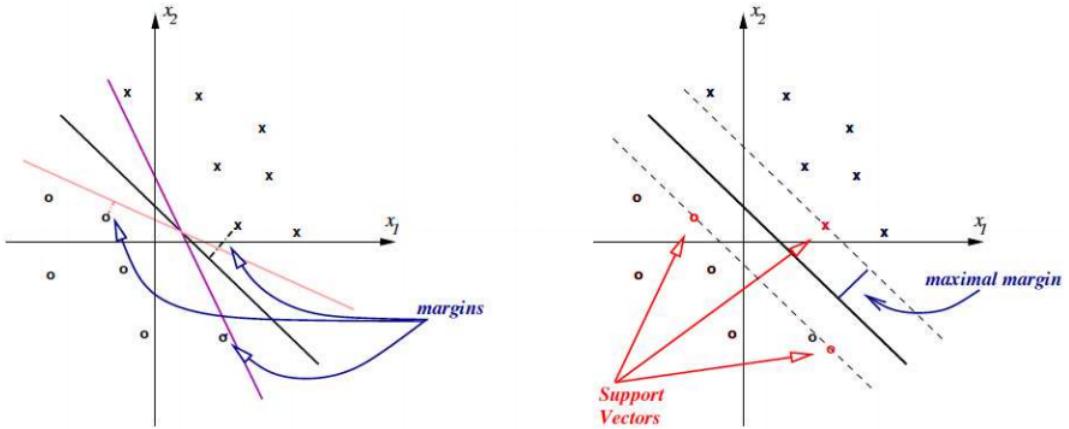
$$\hat{y}(x) = \text{sgn}(w^T x + b)$$

qui classifie LS correctement, c'est-à-dire qui minimise

$$\sum_{k=1}^N \mathbf{1}(y_k \neq \hat{y}(x_k))$$

7.1.1 Hyperplan de marge maximale

Lorsque les données sont linéairement séparables dans l'espace des features, l'hyperplan séparateur n'est pas unique.



Une SVM va chercher à maximiser la distance de l'hyperplan au point le plus proche dans LS, autrement dit

$$\max_{w,b} \underbrace{\min\{\|x - x_k\| : w^T x + b = 0, k = 1, \dots, N\}}_{\text{point le plus proche de la droite}} \underbrace{\gamma}_{\text{maximisation de la marge}}$$

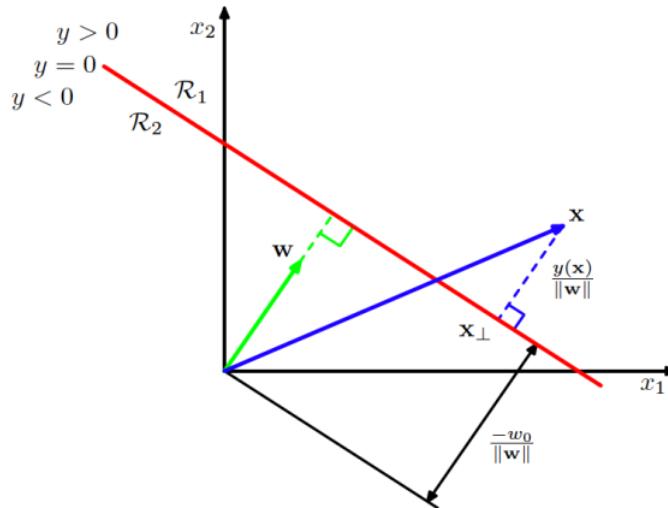
On maximise la marge car, intuitivement, c'est une méthode sûre. De plus, il existe des bornes théoriques sur l'erreur de généralisation qui dépend de la marge :

$$Err(TS) < \mathcal{O}\left(\frac{1}{\gamma}\right)$$

où γ est la marge. Cependant, ces marges ne sont pas souvent atteintes. En pratique, une SVM fonctionne très bien. A noter qu'on a toujours deux points qui sont les plus proches de la droite. C'est toujours le cas, sinon on pourrait encore augmenter la marge.

Cet algorithme conduit à un problème d'optimisation convexe, où la solution peut être écrite uniquement en terme de produits scalaires.

7.1.2 Problème d'optimisation



Supposons un hyperplan séparateur (ligne rouge) et un vecteur de support x . Ce dernier peut s'écrire

$$x = x_\perp + r \frac{w}{\|w\|}$$

où x_\perp est la projection de x sur l'hyperplan et $|r|$ la distance entre x et l'hyperplan. r est donc la marge. En multipliant les deux membres par w^T , on obtient

$$w^T x = \underbrace{w^T x_\perp}_{-b} + r \underbrace{\frac{w^T w}{\|w\|^2}}_{\frac{\|w\|^2}{\|w\|}}$$

car x_\perp est sur
la droite

$$\Leftrightarrow r = \frac{w^T x + b}{\|w\|} = \frac{y(x)}{\|w\|}$$

Le problème d'optimisation qui consiste à maximiser la marge peut alors être écrit comme

$$\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_n [y_n \cdot (w^T x_n + b)] \right\}$$

La solution n'est pas unique vu que l'hyperplan est inchangé si on multiplie w et b par une constante $c > 0$ (par exemple, $c(w^T x + b) = 0$ est aussi une solution).

Pour imposer une unicité et pour normaliser, on choisit typiquement $|w^T x + b| = 1$ pour le point x qui est le plus proche de la surface, autrement dit pour le vecteur de support. On a alors que pour tous les points $k = 1, \dots, N$:

$$y_k(w^T x_k + b) \geq 1$$

Grâce à cette normalisation, la marge vaut désormais $\frac{1}{\|w\|}$, qu'il faut maximiser (ou minimiser $\|w\|$) avec N contraintes

$$\min_{w,b} \varepsilon(w,b) = \frac{1}{2} \|w\|^2$$

sujet aux N contraintes

$$y_k(w^T x_k + b) \geq 1, \forall k = 1, \dots, N$$

$\|w\|$ devient $\frac{1}{2} \|w\|^2$ par facilité (le $\frac{1}{2}$ permet de simplifier lors de la dérivation). Il s'agit d'un problème de programmation quadratique. Il existe une solution seulement si les données sont linéairement séparables, sinon des inéquations ne seront pas satisfaites.

Soient α_k , $k = 1, \dots, N$. On a le lagrangien qui doit être minimisé selon w et b et maximisé selon α :

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{k=1}^N \alpha_k (y_k(w^T x_k + b) - 1)$$

Si on dérive par rapport à w et b , on obtient les conditions optimales suivantes :

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} = 0 &\Leftrightarrow \sum_{j=1}^N \alpha_j y_j x_j = w \\ \frac{\partial \mathcal{L}}{\partial b} = 0 &\Leftrightarrow \sum_{j=1}^N \alpha_j y_j = 0 \end{aligned}$$

Après substitution dans le lagrangien, les conditions optimales conduisent au problème de maximisation dual :

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

sujet aux N inégalités

$$\alpha_k \geq 0, \forall k = 1, \dots, N$$

et à l'égalité

$$\sum_{i=1}^N \alpha_i y_i = 0$$

7.1.3 Vecteurs de support

Le problème primaire est donc

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{k=1}^N \alpha_k (y_k(w^T x_k + b) - 1)$$

En accord avec les conditions complémentaires KKT (Karush-Kuhn-Tucker), le vecteur solution w est tel que

$$\alpha_k (y_k(w^T x_k + b) - 1) = 0, \forall k = 1, \dots, N$$

- $\alpha_k = 0$ si la contrainte est satisfaite comme une inégalité stricte $y_k(w^T x_k + b) > 1$, car c'est la façon de maximiser \mathcal{L} .
 - $\alpha_k > 0$ si la contrainte est satisfaite comme une égalité $y_k(w^T x_k + b) = 1$, auquel cas x_k est le vecteur de support.
- Les seuls points pour lesquelles les contraintes du lagrangien sont actives sont donc les points tels que $y_k(w^T x_k + b) = 1$, autrement dit les vecteurs de support : seuls eux définissent l'hyperplan optimal.

Une fois que les valeurs optimales de α ont été déterminées, le modèle final peut être écrit comme

$$\hat{y}(x) = \operatorname{sgn}\left(\sum_{i=1}^N y_i \alpha_i x_i^T x + b\right)$$

où les valeurs de α_k différentes de 0 (strictement positives) correspondent aux vecteurs de support.

b est calculé en exploitant le fait que pour tout $\alpha_k > 0$, on a nécessairement $y_k(w^T x_k + b) - 1 = 0$.

Avoir un petit ensemble de vecteurs de support est efficace, vu que seuls ces vecteurs x_k , leurs poids α_k et les labels de classe y_k doivent être stockés pour classer de nouveaux exemples.

Si un vecteur non-support x' est retiré de l'échantillon d'apprentissage ou déplacé (en dehors de la région de la marge), la solution est inchangée. De même, elle ne change pas si on ajoute des points qui ne sont pas des vecteurs de support : les SVMs sont peu sensibles aux changements du LS.

La proportion de vecteurs de support dans l'ensemble d'échantillon donne une borne sur l'erreur du leave-one-out :

$$\text{Err}_{\text{loo}} \leq \frac{|\{k | \alpha_k > 0\}|}{N} = \frac{\# \text{ vecteur de support}}{N}$$

7.1.4 Rappel du lagrangien

Supposons que l'on veuille minimiser une fonction $f(x) \in \mathbb{R}$ avec $x \in \mathcal{R}^n$ sujette aux égalités

$$e_i(x) = 0, i = 1, \dots, p$$

et aux inégalités

$$i_j(x) \leq 0, j = 1, \dots, r$$

Le lagrangien est défini par

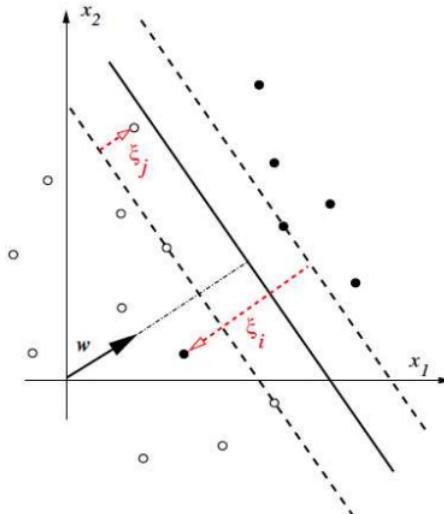
$$\mathcal{L}(x, \alpha, \beta) = f(x) + \alpha^T e(x) + \beta^T i(x), \alpha \in \mathbb{R}^p, \beta \in \mathbb{R}^r$$

Aux optima, les conditions suivantes sont vérifiées :

$$\begin{aligned} \Delta_x \mathcal{L} = 0 &\rightarrow \Delta_x f(x) = \alpha^T \Delta_x e(x) + \beta^T \Delta_x i(x) = 0 \\ \Delta_\alpha \mathcal{L} = 0 &\rightarrow e_i(x) = 0, \forall i = 1, \dots, p \\ \Delta_\beta \mathcal{L} \leq 0 &\rightarrow i_j(x) \leq 0, \beta_j i_j(x) = 0, \beta_j \geq 0, \forall j = 1, \dots, r \end{aligned}$$

7.1.5 Soft margin

A cause du bruit ou de données isolées (outliers), les données peuvent ne pas être linéairement séparables dans l'espace des features.



Les discordances sont mesurées par les variables $\xi_i \geq 0$ avec la contrainte relaxée associée $y_i(w^T x_i + b) \geq 1 - \xi_i$. En rendant ξ_i suffisamment large, la contrainte peut toujours être satisfaite :

- si $0 < \xi_i < 1$, la marge n'est pas satisfaite mais x_i est toujours correctement classé
- si $\xi_i > 1$, alors x_i est mal classé.

Marge douce en norme 1

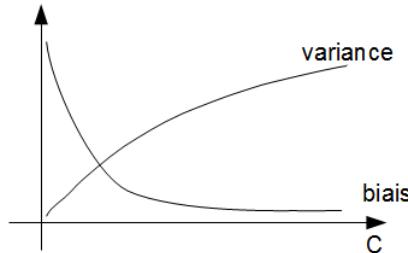
Le problème primal est

$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

s.t.

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i = 1, \dots, N$$

où C est une constante positive et qui équilibre l'objectif de maximiser la marge et de minimiser l'erreur engendrée. Ainsi, si $C = 0$, on ne pénalise pas les ξ_i donc les mauvais classements. Plus C est grand et plus on augmente la complexité, car les α_k peuvent être grands et on veut un meilleur classement (donc on prend de plus en plus en compte les ξ_i).



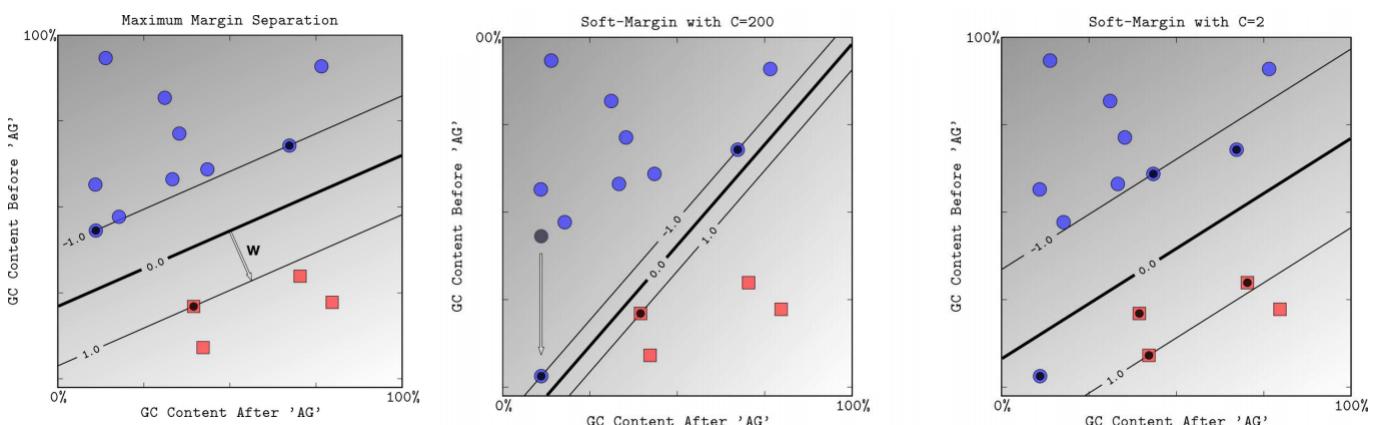
On a le problème dual :

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

s.t.

$$0 \leq \alpha_k \leq C, \forall k = 1, \dots, N \text{ (contraintes de boîte)}$$

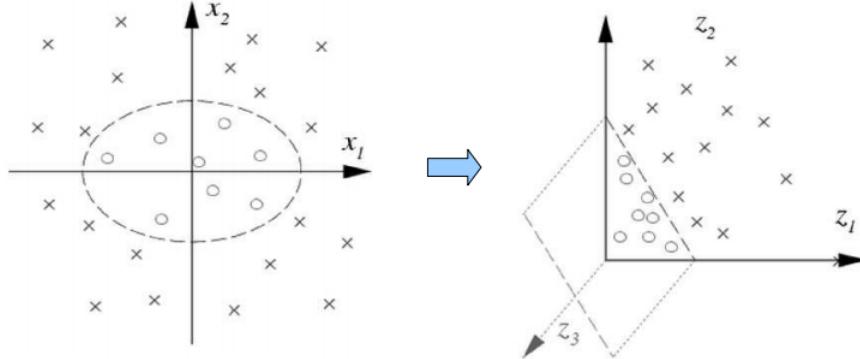
$$\sum_{i=1}^N \alpha_i y_i = 0$$



Les points dans la marge correspondent à des $\xi_i > 0$. La figure du milieu illustre du sur-apprentissage.

7.2 Noyaux dans les SVMs

Les données d'apprentissage peuvent ne pas être linéairement séparables dans l'espace d'entrée. On considère alors un mapping non linéaire ϕ vers un nouvel espace de features.



L'hyperplan est donc défini par

$$y(x) = w^T \phi(x) + b$$

Le problème dual devient

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

Plutôt que de définir le mapping ϕ , on peut directement définir le produit scalaire $K(x, x') = \phi(x)^T \phi(x')$, ce qui rend le mapping implicite.

Il est possible de caractériser mathématiquement les fonctions $K(x, x')$ définies sur des paires d'objets : cette fonction est appelée un noyau (positif ou de Mercer). Il peut cependant être difficile de trouver une mesure de la similarité entre x et x' .

L'hyperplan séparateur est alors défini par

$$y(x) = \sum_{k=1}^N \alpha_k y_k K(x_k, x) + b$$

Le modèle devient alors

$$\hat{y}(x) = \operatorname{sgn}\left(\sum_{k=1}^N y_k \alpha_k K(x, x_k) + b\right)$$

où les α_k peuvent être déterminés en résolvant le problème de maximisation quadratique

$$\max_{\alpha} \mathcal{W}(\alpha) = \sum_{k=1}^N \alpha_k - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

sujet aux N contraintes d'inégalité

$$\alpha_k \geq 0, \forall k = 1, \dots, N$$

et la contrainte d'égalité

$$\sum_{i=1}^N \alpha_i y_i = 0$$

7.3 Kernel trick

Le kernel trick consiste à transformer l'espace d'entrée en un autre espace où l'algorithme se comporte mieux/est plus rapide. N'importe quel algorithme qui utilise les données avec uniquement des produits scalaires peut se baser sur ce mapping implicite, en remplaçant $x^T x'$ par $K(x, x')$.

Les avantages sont que

- on a un calcul moins coûteux qu'un produit scalaire en grande dimension
- la transformation ϕ est implicite

7.4 Notion mathématique du noyau

Soit U un ensemble d'objets non vide. Un noyau positif est une fonction $K(.,.)$

$$K(.,.) : U \times U \rightarrow \mathbb{R}$$

telle que pour tout $N \in \mathbb{N}$ et pour tout $o_1, \dots, o_N \in U$, la matrice $N \times N$

$$K : K_{i,j} = K(o_i, o_j)$$

est symétrique et définie semi-positive, c'est-à-dire que ses valeurs propres sont positives, ou bien que $z^T K z \geq 0, \forall z \in \mathbb{R}^n$.

Pour tout noyau positif K défini sur U , il existe un espace de produit scalaire \mathcal{V} et une fonction $\phi(.) : U \rightarrow \mathcal{V}$ tel que

$$K(o, o') = \phi(o) \times \phi(o')$$

où l'opérateur \times dénote le produit scalaire dans \mathcal{V} .

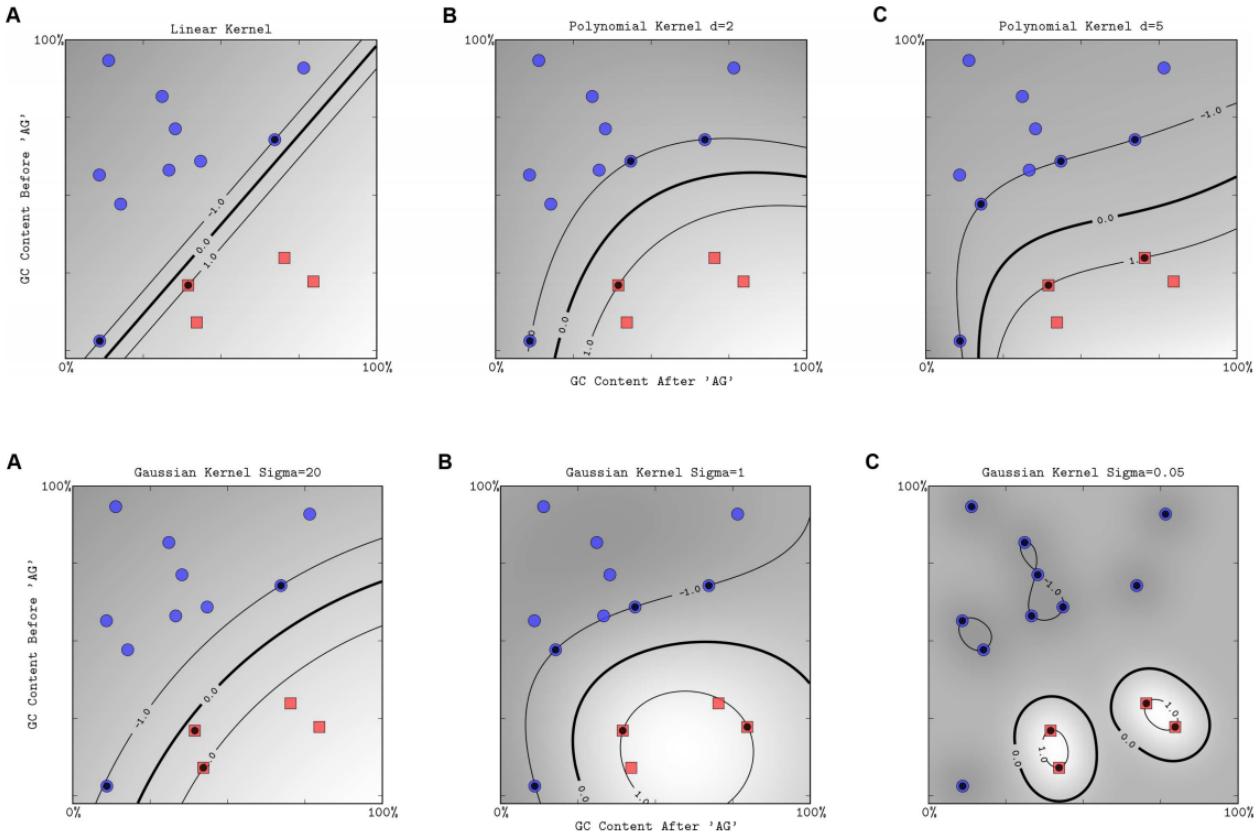
En général, l'espace \mathcal{V} n'est pas nécessairement de dimension finie.

Le noyau définit un produit scalaire, et donc une norme et une mesure de distance sur U , qui est héritée de \mathcal{V} :

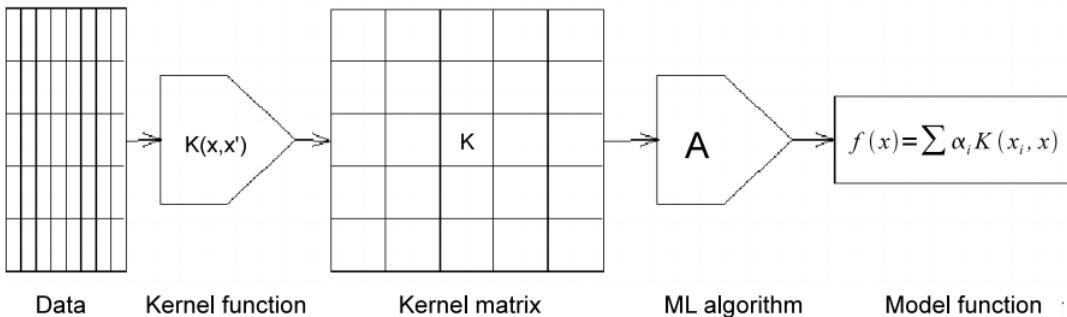
$$\begin{aligned} d_U^2(o, o') &= d_{\mathcal{V}}^2(\phi(o), \phi(o')) \\ &= (\phi(o) - \phi(o'))^T (\phi(o) - \phi(o')) \\ &= \phi(o) \times \phi(o) + \phi(o') \times \phi(o') - 2\phi(o) \times \phi(o') \\ &= K(o, o) + K(o', o') - 2K(o, o') \end{aligned}$$

7.5 Exemple de noyaux

- noyau constant : $K(o, o') = 1$
- noyau linéaire défini sur des attributs numériques : $K(o, o') = \mathbf{a}^T(o)\mathbf{a}(o') = \langle \mathbf{a}(o), \mathbf{a}(o') \rangle$ (produit scalaire)
- noyau polynomial : si d est le degré maximum $K(o, o') = (\langle \mathbf{a}(o), \mathbf{a}(o') \rangle + 1)^d$
- noyau de Hamming pour des attributs discrets : $K(o, o') = \sum_{i=1}^m \delta_{\mathbf{a}_i(o), \mathbf{a}_i(o')}$ (nombre de composantes communes aux deux vecteurs)
- noyau de texte, qui calcule le nombre de sous-chaînes communes dans o et o' (dimension infinie si la taille du texte n'est à priori pas bornée)
- combinaison de noyaux :
 - la somme de plusieurs noyaux (positifs) est toujours un noyau (positif)
 - le produit de plusieurs noyaux est aussi un noyau
 - noyaux polynomiaux : $\sum_{i=0}^n a_i (K(x, x'))^i$ si $\forall i : a_i \geq 0$
 - $K(x, x') = (x^T x')^2$ (voir slide 31/45 pour plus de détails)
 - noyau gaussien/à base radiale : si σ est l'étalement de la distribution, $K(x, x') = \exp \frac{-(x-x')^T(x-x')}{2\sigma^2}$



7.6 Méthodes à noyaux



L'approche est modulaire : on découpe l'algorithme de la représentation. Beaucoup d'algorithmes peuvent utiliser des noyaux : ridge regression, PCA, k-means, etc. Des noyaux ont été définis pour plusieurs types de données : séries temporelles, images, graphes, séquences, etc.

Les intérêts principaux des noyaux est que l'on peut travailler efficacement dans des espaces de dimension très élevée (potentiellement infinie) dès qu'un produit scalaire est facile à calculer, et qu'on peut appliquer des algorithmes classiques sur des données en toute généralité, sans être nécessairement vectorielles (par exemple construire un modèle linéaire sur un graphe).

7.7 Forces et faiblesses des SVMs

- + les SVMs sont motivées théoriquement
- + classificateur des plus efficaces
- + implémentations efficaces pour des problèmes larges
- modèles en boîte noire
- le choix d'un bon noyau est difficile et critique pour atteindre une bonne précision

L'optimisation convexe est un outil très utile en apprentissage, car beaucoup de problèmes peuvent être formulés comme des problèmes d'optimisation convexe. On bénéficie également du travail important donné pour créer des algorithmes d'optimisation efficaces, et qui plus est donnent une solution unique (ce qui rend le problème plus stable).

Chapitre 8

Méthodes d'ensemble

Pour une méthode donnée, il y a généralement un trade-off à faire entre le biais et la variance. Il est possible d'améliorer le modèle (par exemple avec du pruning pour les arbres de décision), mais ce n'est pas toujours possible. Le but des méthodes d'ensemble est de modifier ce trade-off, quitte à perdre des features de la méthode initiale.

L'idée est de combiner plusieurs modèles construits avec un algorithme d'apprentissage afin d'améliorer la précision. Les arbres de décision sont souvent utilisés pour des raisons d'efficacité.

Il existe deux familles de méthode :

- les techniques de moyennage, où la prédiction finale n'est que la moyenne des prédictions, et qui permettent de surtout faire diminuer la variance.
- les algorithmes de boosting, où on crée des modèles séquentiellement afin de diminuer le biais.

8.1 Bagging

8.1.1 Approche théorique

Supposons que l'on puisse générer plusieurs échantillons d'apprentissages $\{\text{LS}_1, \text{LS}_2, \dots, \text{LS}_T\}$ à partir de la distribution des données originales $P(\underline{x}, y)$. Pour chacun des LS_i , on va apprendre un modèle \hat{y}_{LS_i} et on va calculer la moyenne :

$$\hat{y}_{\text{ens}} = \frac{1}{T} \sum_{i=1}^T \hat{y}_{\text{LS}_i}(\underline{x})$$

Pour rappel, on a l'erreur moyenne suivante :

$$\begin{aligned} E_{\text{LS}}\{\text{Err}(\underline{x})\} &= \mathbb{E}_{y|\underline{x}}\{(y - h_B(\underline{x}))^2\} \\ &+ (h_B(\underline{x}) - E_{\text{LS}}\{\hat{y}(\underline{x})\})^2 \\ &+ E_{\text{LS}}\{(\hat{y}(\underline{x}) - E_{\text{LS}}\{\hat{y}(\underline{x})\})^2\} \end{aligned}$$

Le biais n'est pas différent par rapport à l'algorithme original. En effet,

$$\mathbb{E}_{\text{LS}_1, \dots, \text{LS}_T}\{\hat{y}_{\text{ens}}(\underline{x})\} = \frac{1}{T} \sum_i \mathbb{E}_{\text{LS}_i}\{\hat{y}_{\text{LS}_i}(\underline{x})\} = E_{\text{LS}}\{\hat{y}_{\text{LS}}(\underline{x})\}$$

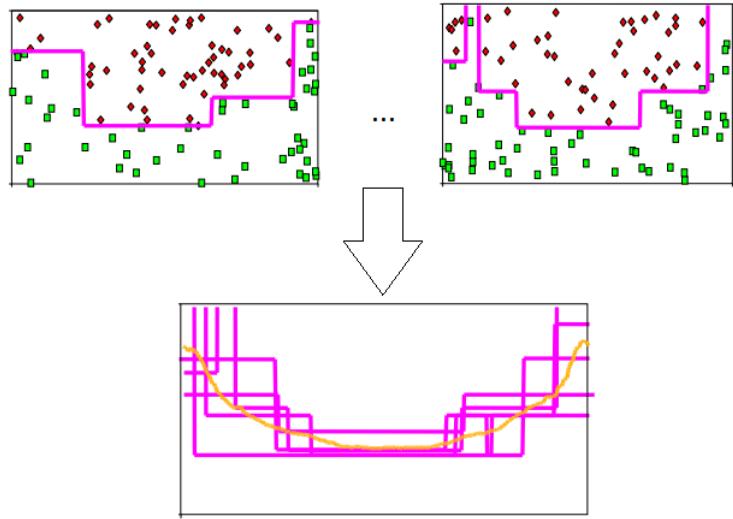
Par contre, la variance est divisée par T :

$$\mathbb{E}_{\text{LS}_1, \dots, \text{LS}_T}\{(\hat{y}_{\text{ens}}(\underline{x}) - \mathbb{E}_{\text{LS}_1, \dots, \text{LS}_T}\{\hat{y}_{\text{ens}}(\underline{x})\})^2\} = \frac{1}{T} E_{\text{LS}}\{(\hat{y}(\underline{x}) - E_{\text{LS}}\{\hat{y}_{\text{ens}}(\underline{x})\})^2\}$$

En effet, différents échantillons d'apprentissage conduisent à différents modèles, surtout si l'algorithme sur-apprend les données. Vu qu'il n'y a qu'un seul modèle optimal, la variance est la source d'erreur.

8.1.2 En pratique

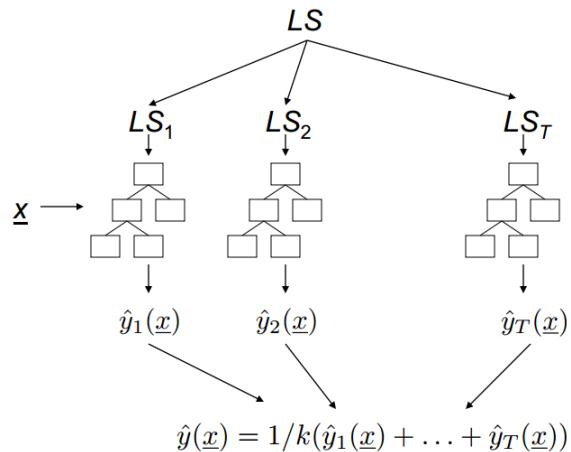
Il n'est généralement pas possible de créer plusieurs LS, car $P(\underline{x}, y)$ est inconnu, et les méthodes d'ensembles nécessitent justement beaucoup de données. L'idée est d'utiliser du bootstrap sampling pour générer plusieurs ensembles d'apprentissage.



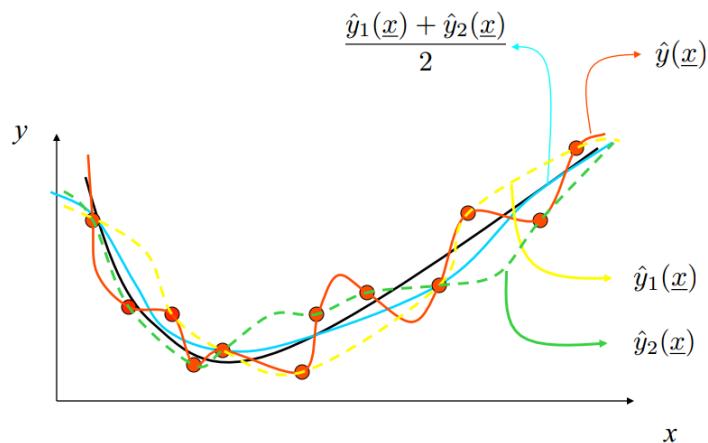
On a alors l'algorithme du bagging (**bootstrap aggregating**) :

- on crée T bootstrap samples $\{B_1, \dots, B_T\}$ à partir de LS
- on apprend un modèle \hat{y}_{B_i} pour chaque B_i
- on construit le modèle de moyenne :

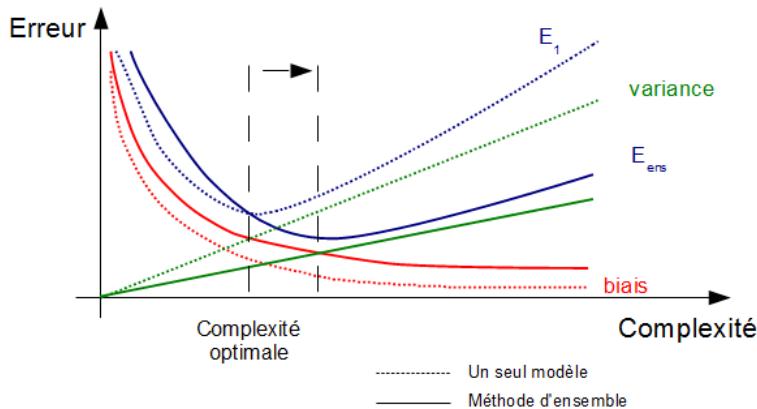
$$\hat{y}_{\text{ens}}(\underline{x}) = \frac{1}{T} \sum_{i=1}^T \hat{y}_{B_i}(\underline{x})$$



Dans le cas d'une classification, $\hat{y}(\underline{x})$ sera la classe majoritaire parmi $\{\hat{y}_1(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$



La variance est réduite, mais le biais augmente un peu car la taille effective du bootstrap sample est environ 30% plus petite que le LS original ; tous les exemples de départ ne s'y trouvent pas, seulement environ ~ 63.2%.



L'erreur est moindre, par contre il faut souvent augmenter la complexité pour obtenir l'erreur minimale.

8.1.3 Autres techniques de moyennage

Paradigme de perturbation et combinaison : on perturbe les données ou l'algorithme d'apprentissage pour obtenir plusieurs modèles qui sont bons sur l'échantillon d'apprentissage, puis on combine les prédictions des modèles.

La variance est généralement diminuée (grâce à la moyenne), mais le biais augmente un peu à cause de la perturbation.

Exemples : le bagging perturbe l'échantillon d'apprentissage, les réseaux de neurones peuvent être initialisés avec des poids aléatoires, random forests.

8.2 Random Forests

C'est un algorithme de type *perturb and combine* conçu spécifiquement pour les arbres. Elle utilise du bagging et une sélection aléatoire d'un ensemble d'attributs. L'algorithme suivant est utilisé :

- on construit l'arbre sur un bootstrap sample
- au lieu de choisir le meilleur split sur tous les attributs (au nombre de p), on sélectionne le meilleur split parmi un sous-ensemble de k attributs

Il y a un trade-off biais/variance avec k : plus k est petit et plus la réduction est grande, mais plus haut sera le biais. Faire la moyenne casse le compromis biais/variance habituel.

Le fait d'effectuer des choix aléatoires fait que l'arbre ne dépend pas de l'échantillon d'apprentissage.

Method	E	Bias	Variance
Full regr. Tree	10.2	3.5	6.7
Bagging ($k=10$)	5.3	3.8	1.5
Random Forests ($k=7$)	4.8	3.8	1.0
Random Forests ($k=5$)	4.9	4.0	0.9
Random Forests ($k=3$)	5.6	4.7	0.8

Un autre avantage des random forests est de diminuer le temps de calcul par rapport au bagging, car seul un sous-ensemble d'attributs est considéré lorsqu'on split un noeud (et pas tous). Généralement, $k = \sqrt{p}$.

8.3 Décomposition de l'ambiguïté

Supposons que l'on ait T modèles $\{\hat{y}_1, \dots, \hat{y}_T\}$ et leur moyenne

$$\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_i \hat{y}_i(\underline{x})$$

$$\begin{aligned}
\frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}(\underline{x}))^2\} &= \frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_{\text{ens}}(\underline{x}) + \hat{y}_{\text{ens}}(\underline{x}) - \hat{y}_i(\underline{x}))^2\} \\
&= \frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_{\text{ens}}(\underline{x}))^2\} \\
&+ \frac{1}{T} \sum_i \underbrace{\mathbb{E}_{y|\underline{x}} \{(\hat{y}_{\text{ens}}(\underline{x}) - \hat{y}_i(\underline{x}))^2\}}_{\star_1} \\
&+ \underbrace{\frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y(\underline{x}) - \hat{y}_{\text{ens}}(\underline{x})) (\hat{y}_{\text{ens}}(\underline{x}) - \hat{y}_i(\underline{x}))\}}_{\mathbb{E}_{y|\underline{x}} \{y - \hat{y}_{\text{ens}}(\underline{x})\} \frac{1}{T} \sum_i (y_{\text{ens}}(\underline{x}) - \hat{y}_i(\underline{x})) \star_2} \\
&\quad \text{ne dépend pas de } i \quad \text{ne dépend pas de } y
\end{aligned}$$

\star_1 : on peut supprimer la somme car l'intérieur ne dépend pas de y .

\star_2 : $y_{\text{ens}}(\underline{x})$ ne dépend pas de i , donc on peut faire rentrer la somme et annuler le terme car $\sum_i \hat{y}_i(\underline{x}) = y_{\text{ens}}(\underline{x})$.
Au final, on a

$$\begin{aligned}
\frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_i(\underline{x}))^2\} &= \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_{\text{ens}}(\underline{x}))^2\} + \frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{\text{ens}}(\underline{x}))^2 \\
\Leftrightarrow \underbrace{\mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_{\text{ens}}(\underline{x}))^2\}}_{\text{Erreur de la méthode d'ensemble}} &= \underbrace{\frac{1}{T} \sum_i \mathbb{E}_{y|\underline{x}} \{(y - \hat{y}_i(\underline{x}))^2\}}_{\text{Moyenne des erreurs des modèles}} - \underbrace{\frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{\text{ens}}(\underline{x}))^2}_{\text{Ambiguïté de la méthode d'ensemble}}
\end{aligned}$$

L'ambiguïté mesure la variabilité des prédictions des modèles individuels. A moins que tous les modèles ne soient les mêmes, ce terme sera toujours positif. Du coup, plus les sous-modèles sont diversifiés et plus l'ambiguïté sera grande, donc plus la méthode d'ensemble sera efficace.

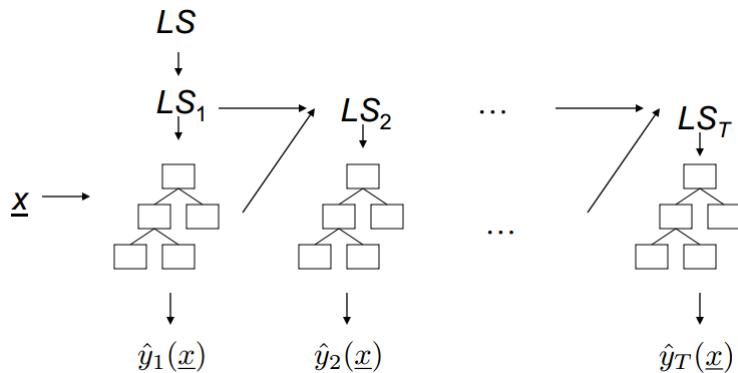
Le modèle moyen est donc toujours meilleur que les modèles individuels en moyenne. Ce n'est cependant pas vrai en classification.

8.4 Boosting

L'idée est de combiner plusieurs modèles "faibles", afin de produire un modèle plus puissant. Un modèle est considéré comme faible s'il a un grand biais (en classification, si le modèle est à peine meilleur qu'un choix aléatoire).

Les différences par rapport aux autres méthodes d'ensemble sont que

- les modèles sont construits séquentiellement sur des versions modifiées des données
- les prédictions des modèles sont combinées à travers une somme pondérée/un vote



Par exemple, pour passer de LS_1 à LS_2 , on ignore les points bien classés de LS_1 .

En régression,

$$\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \cdots + \beta_T \hat{y}_T(\underline{x})$$

En classification, $\hat{y}(\underline{x})$ sera la classe majoritaire dans $\{\hat{y}_1(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$, en tenant compte des points $\{\beta_1, \dots, \beta_T\}$.

8.4.1 Adaboost

On suppose que l'algorithme d'apprentissage utilisé accepte les objets pondérés : $\{(x_1, y_1, w_1), \dots, (x_N, y_N, w_N)\}$. C'est le cas de beaucoup d'algorithmes ; avec les arbres, on prend en compte les poids quand on compte les objets ; avec un réseau de neurones, on minimise l'erreur au carré pondérée.

A chaque étape, Adaboost augmente les poids dans le cas où l'échantillon est mal classé par le modèle, ainsi l'algorithme se focalise sur les cas compliqués de l'échantillon d'apprentissage. β_i sera ainsi plus petit si le modèle commet beaucoup d'erreurs. Par exemple,

- si l'exemple est mal classé, $w_i \leftarrow w_i \exp \beta$
- si l'exemple est bien classé, $w_i \leftarrow w_i \cdot 1$

On a l'algorithme suivant : il prend en entrée un algorithme d'apprentissage et un échantillon d'apprentissage $\{(x_i, y_i) : i = 1, \dots, N\}$. On initialise les poids $w_i = \frac{1}{N}$, $i = 1, \dots, N$.

Pour $t = 1$ jusque T :

- on construit un modèle $\hat{y}_t(x)$ avec l'algorithme d'apprentissage en utilisant les poids w_i
- on calcule l'erreur pondérée :

$$\text{Err}_t = \frac{\sum_i w_i \cdot I(y_i \neq \hat{y}_t(x_i))}{\sum_i w_i}$$

- on calcule $\beta_t = \log \frac{1 - \text{Err}_t}{\text{Err}_t}$
- on met à jour les poids :

$$w_i \leftarrow w_i \cdot \exp \beta_t \cdot I(y_i \neq \hat{y}_t(x_i))$$

- on normalise les poids, de façon à ce que $\sum_i w_i = 1$

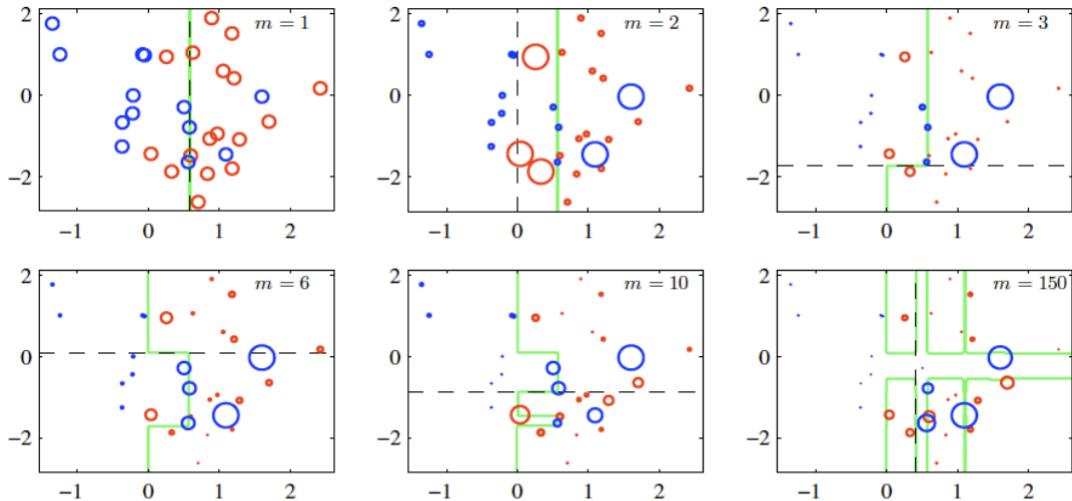


Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner.

8.4.2 Boosting aux moindres carrés

Cet algorithme de boosting est conçu pour les régressions. Il prend en entrée un algorithme d'apprentissage et un échantillon d'apprentissage $\{(x_i, y_i) : i = 1, \dots, N\}$. On initialise de la façon suivante :

$$\hat{y}_0(x) = \frac{1}{N} \sum_i y_i$$

$$r_i = y_i, \quad i = 1, \dots, N$$

Pour $t = 1$ jusque T :

- pour $i = 1$ jusque N , on calcule les résiduels :

$$r_i \leftarrow r_i - \hat{y}_{t-1}(x_i)$$

- on construit un arbre de régression sur l'échantillon d'apprentissage

$$\{(x_i, r_i) : i = 1, \dots, N\}$$

On retourne ensuite le modèle définit par

$$\hat{y}(x) = \hat{y}_0(x) + \hat{y}_1(x) + \dots + \hat{y}_T(x)$$

8.4.3 Algorithme de boosting générique

Le but est de trouver $\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \dots + \beta_T \hat{y}_T(\underline{x})$ et qui minimise $\sum_{i=1}^N L(y_i, \hat{y}(x_i))$.
On utilise du *forward stage-wise additive modeling* :

1. On initialise $\hat{y}(\underline{x}) = 0$
2. pour $t = 1$ jusque T :
 - (a) on calcule

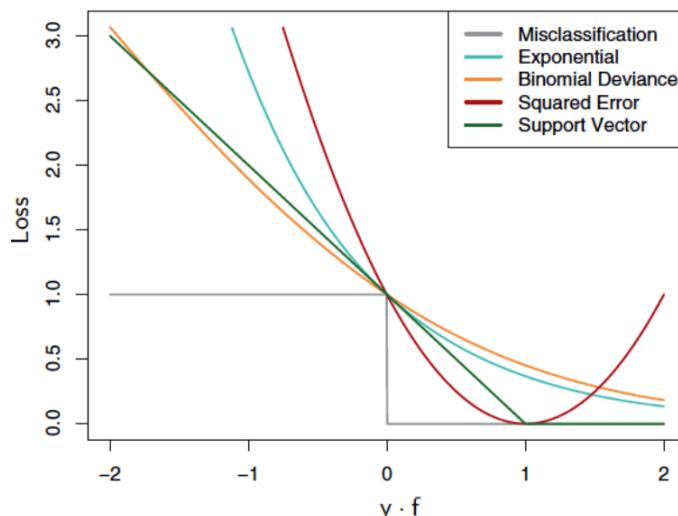
$$(\beta_t, \hat{y}_t) = \arg \min_{\beta, \hat{y}'} \sum_i L(y_i, \hat{y}(\underline{x}_i) + \beta \hat{y}'(\underline{x}_i))$$

(b)

$$\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \beta_t \hat{y}_t(\underline{x})$$

On a ainsi

- pour le boosting aux moindres carrés, $L(y, y') = (y - y')^2$
- pour Adaboost, $L(y, y') = \exp(-yy')$



La courbe rouge (squared error) n'est pas intéressante, car elle pénalise les modèles où $y.f > 1$, or il faudrait qu'elle tende vers 0 comme les autres.

8.4.4 LS boosting

Remarque : Je n'ai pas retrouvé dans les transparents quoi que ce soit qui ait un rapport à cette section. Cela vient de mes notes, mais elles sont assez lacunaires, du coup ce qui suit ne veut pas dire grand chose.

On a un paramètre $u \in [0, 1]$ qui permet de ralentir l'évolution de l'algorithme, et donc le sur-apprentissage.

x_1	\dots	x_p	y	y_0	y_1
y_1			$y_1 - u\bar{y}$?	
y_2			$y_2 - u\bar{y}$?	
\vdots			\vdots		?

Donc, à chaque itération i , on utilise ce qui a été obtenu à l'itération $i - 1$ avec un vecteur u , comme on ferait avec une moyenne exponentielle.

8.4.5 Autres méthodes de boosting

Il existe de nombreux autres algorithmes de boosting (par exemple le gradient boosting).

Le boosting sur les arbres de décision/régression améliore très fortement leur précision, cependant on est beaucoup plus sensible au bruit (sur-apprentissage) que les techniques de moyennage.

Pour que le boosting fonctionne, il faut que les modèles ne soient pas parfaits sur l'échantillon d'apprentissage. Pour les arbres, on peut soit les élaguer (pre-prune ou post-prune avec de la validation croisée), soit limiter le nombre de tests (et spliter d'abord sur les noeuds les plus impures). Il y a encore une fois un trade-off biais/variance qui dépend de la taille de l'arbre.

8.5 Interprétabilité et efficacité

Lorsque les méthodes d'ensemble sont combinées avec les arbres de décision, ils perdent de l'interprétabilité et de l'efficacité. En revanche, on les utilise toujours pour calculer l'importance des variables, en effectuant la moyenne sur tous les arbres. De plus, les méthodes d'ensemble peuvent être parallélisées et l'algorithme boosting utilise des petits arbres, ce qui fait que le coût en temps processeur n'est pas important.

Method	E	Bias	Variance
Full regr. Tree	10.2	3.5	6.7
Regr. Tree with 1 test	18.9	17.8	1.1
+ MART (T=50)	5.0	3.1	1.9
+ Bagging (T=50)	17.9	17.3	0.6
Regr. Tree with 5 tests	11.7	8.8	2.9
+ MART (T=50)	6.4	1.7	4.7
+ Bagging (T=50)	9.1	8.7	0.4

8.6 Autres approches d'ensemble

8.6.1 Moyenne de modèles de Bayes

$$P(y|x, \text{LS}) = \sum_{h \in \mathcal{H}} P(y|h, \text{LS})P(h|\text{LS})$$

$$P(h|\text{LS}) \propto P(h)P(\text{LS}|h) \propto P(h) \sum_{\theta} P(\text{LS}|\theta, h)P(\theta|h)$$

8.6.2 Stacking

On apprend un modèle qui combine des modèles. Soit $\text{LS} = \{(x_i, y_i) : i = 1, \dots, N\}$ et soit A^T ($t = 0, \dots, T$) $T+1$ algorithmes d'apprentissage.

Pour $t = 1, \dots, T$:

- construire un modèle :

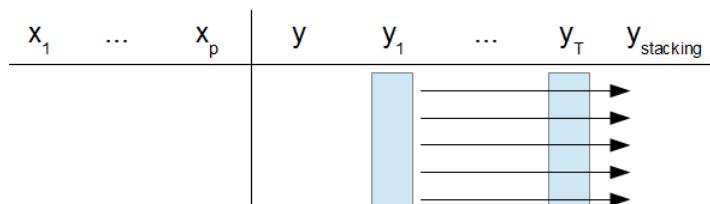
$$\hat{y}^t = A^t(\text{LS})$$

- calculer les prédictions :

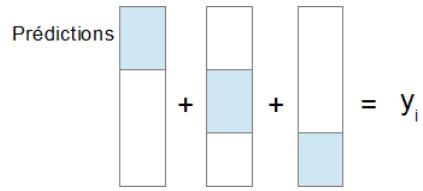
$$\hat{y}_i^t = \hat{y}^t(x_i)$$

Soit $\text{LS}^0 = \{(x_i^0, y_i)\}$ avec $x_i^0 = (y_i^t)_{t=1}^T$: on retourne $\hat{y} = A^0(\text{LS}^0)$.

Il s'agit d'une généralisation du boosting, dont le modèle est une combinaison linéaire des sous-modèles. L'algorithme d'apprentissage s'entraîne sur les sorties des sous-modèles.



Lors de la phase d'apprentissage des sous-modèles, il faut que les y_i soient différents, par exemple avec de la validation croisée.



8.7 Conclusion

Les méthodes d'ensemble sont très efficaces pour réduire le biais et/ou la variance, en transformant une méthode pas si bonne en une méthode très compétitive. Adaboost avec des arbres est considéré comme un des meilleurs algorithmes de classification.

L'interprétabilité et l'efficacité du modèle sont difficiles à préserver si on veut réduire la variance significativement.

Chapitre 9

Sélection de features

Réduire le nombre de variable est utile car

- on évite le sur-apprentissage (car on apprend sur des variables inutiles) et on améliore les performances du modèle
- on améliore l'interprétabilité
- les modèles sont plus rapides et moins coûteux
- on réduit tous les temps de calcul (si la sélection est elle-même rapide)

Il existe deux familles de méthodes :

- la sélection de features : on cherche un petit (ou le plus petit) sous-ensemble de features qui maximisent la précision du modèle
- le ranking de features : on trie les variables selon leur importance à la prédiction de la sortie

A noter qu'on peut obtenir une sélection de feature avec un ranking : il suffit de sélectionner les k premières features.

Trois approches existent dans la sélection de features :

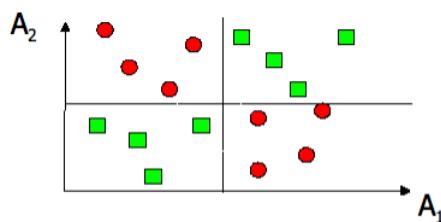
- filtre : sélection à-priori des variables, indépendamment d'un algorithme d'apprentissage supervisé
- embarqué (embedded) : sélection de features embarquée dans un algorithme d'apprentissage
- wrapper : utiliser la validation croisée pour trouver l'ensemble optimal de features pour un algorithme

9.1 Techniques de filtre

L'idée est d'associer un score à chaque feature et de retirer celles dont le score est trop bas.

Généralement, on utilise une fonction de score univariable, comme avec les arbres de décision ou avec des tests statistiques (t-test, chi², etc). Le nombre optimal de features peut être déterminé par de la validation croisée.

Des approches de sélection multi-variable existent (dans les arbres de décision, etc), et peuvent être utiles ; des features pourraient être inutiles toutes seules (on a un petit score univariable), mais ensembles, elles peuvent parfaitement expliquer la classification.



Avantages et inconvénients :

- + l'approche univariable est rapide et scalable
- + on est indépendant de l'algorithme d'apprentissage supervisé
- on ignore l'algorithme d'apprentissage, et donc potentiellement une bonne combinaison de variables
- l'approche univariable ignore la dépendance entre les features
- l'approche multivariable est beaucoup plus lente

9.2 Techniques embarquées

Certains algorithmes d'apprentissage supervisé embarquent une sélection de features ; la recherche d'un sous-ensemble optimal de features se fait dans l'algorithme même.

Exemples :

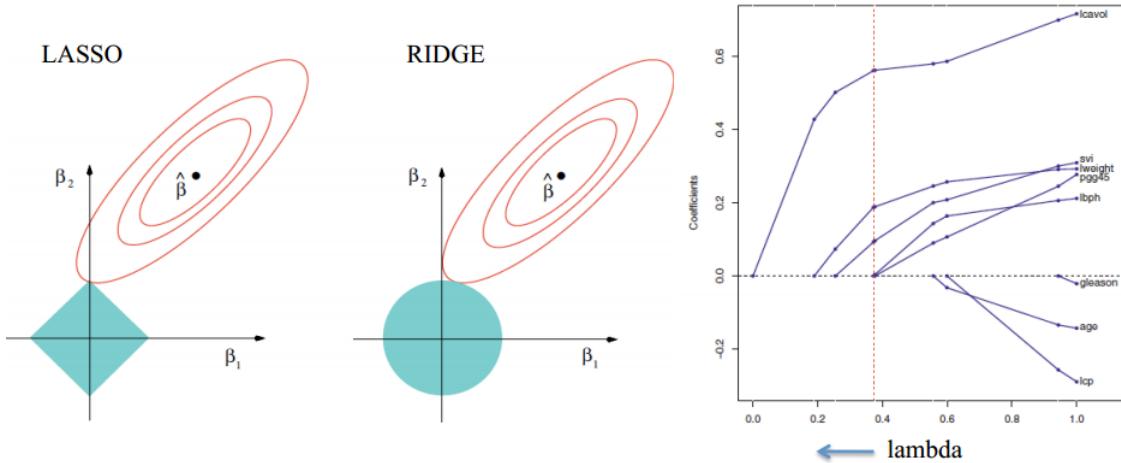
- le splitting des arbres de décision
 - les méthodes d'ensemble à base d'arbres
 - les valeurs absolues des poids dans une SVM linéaire : $\hat{y}(\underline{x}) = \text{sgn}(\sum_i w_i x_i + b)$
- Avantages et inconvénients :
- + généralement efficace en terme de temps de calcul
 - + bonne intégration avec l'algorithme d'apprentissage
 - + multivariable
 - spécifique à l'algorithme d'apprentissage

9.2.1 LASSO

Il s'agit d'un modèle linéaire avec une pénalisation L1, comme de la ridge regression ; le carré est remplacé par une valeur absolue.

$$\min_{\beta} \sum_{i=1}^N (y_i - (\beta_0 + \sum_j \beta_j x_j))^2 + \lambda \sum_j |\beta_j|$$

Les courbes rouges représentent la valeur des premiers termes, sans régularisation.



Le problème est identique à

$$\begin{aligned} & \min_{\beta} \sum_i (y_i - \beta \underline{x}_i)^2 \\ & \text{tel que } \sum_j |\beta_j| < C \text{ (losange)} \\ & \text{ou } \sum_j \beta_j^2 < C \text{ (cercle)} \end{aligned}$$

Un losange est mieux car on a plus de chance d'arriver au sommet d'un losange qu'au sommet d'un cercle, donc d'avoir un β_i nul. Pour $|\beta|^p$, si $p \leq 1$, on se dirige vers une sorte d'étoile à quatre branches (\diamond), ce qui augmente encore plus la probabilité.

Si λ augmente, les coefficients rétrécissent ; on obtient un classement des variables (celle qui dure le plus longtemps est la plus importante).

Cette méthode est plus lente que la *ridge regression* car il y a un problème d'optimisation, et on est limité aux méthodes linéaires.

9.3 Techniques de wrapper

On essaie de trouver le sous-ensemble de features qui maximise la qualité du modèle induite par l'algorithme d'apprentissage. Cette qualité est estimée par de la validation croisée. Vu que le nombre de sous-ensemble d'un ensemble de p features est de 2^p , tous les sous-ensembles ne sont pas évalués et des heuristiques sont nécessaires.

Plusieurs approches existent :

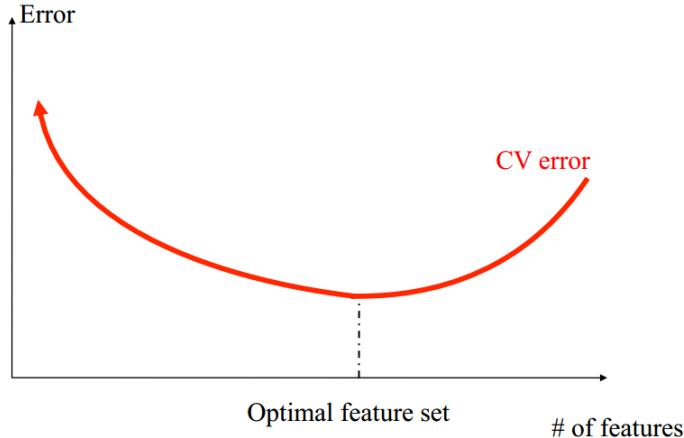
- forward (ou backward) selection : on ajoute (retire) les variables qui diminuent le plus (augmentent le moins) l'erreur

- optimisation avec des algorithmes génétiques

Une approche populaire est l'élimination récursive de features. On suppose que l'on dispose d'un algorithme d'apprentissage qui peut ranker les features (ex : SVM, arbres de décision). A partir de l'ensemble complet de features, on itère :

- on apprend un modèle à partir de l'ensemble courant de features
- on calcule le rang des features avec le modèle
- on retire la feature avec le plus petit rank

Au final, on garde l'ensemble de features qui donne la plus petite erreur par validation croisée.



Avantage et inconvénients :

- + l'ensemble de features est taillé sur mesure pour l'algorithme d'apprentissage
- + il est possible de trouver des interactions et de supprimer des variables redondantes
- méthode susceptible au sur-apprentissage : il est parfois facile de trouver un petit sous-ensemble de variables bruitées qui donne de très bons résultats
- coûteux, car il faut construire un modèle sur chaque sous-ensemble de variables

9.4 Biais de sélection

Mauvaise méthode de sélection A partir de la base de données, on sélectionne les N meilleures variables avec un filtre. On évalue ensuite un algorithme qui utilise ces N variables par validation croisée sur la base de données.

Supposons une expérience artificielle où les variables sont choisies aléatoirement, de même que la classe de sortie. On effectue alors deux essais :

- tree bagging sans sélection de feature : erreur en validation croisée 10-fold = 52%
- idem mais avec les 20 meilleures features : 35%

On pourrait supposer qu'il y a 20 variables intéressantes et qu'avec elles on pourrait créer un meilleur classificateur qu'un classificateur aléatoire. Cependant, sur un nouvel ensemble d'échantillons, on obtient une erreur de 52%. De plus, le 35% obtenu n'est pas possible, car toutes les valeurs sont aléatoires : tout modèle aura une erreur de 50%.

On a un problème de sur-apprentissage, car on a sélectionné et testé le modèle sur base de toute la base de données. Le bon protocole aurait dû être

- diviser LS en 10 sous-ensembles
- pour $i = 1$ jusqu'à 10 :
 - retirer le i ème sous-ensemble de LS
 - sélectionner les 20 variables sur les sous-ensembles restant
 - apprendre le modèle sur les 20 variables et les objets restant
 - tester le modèle sur le i ème sous-ensemble de variables

Il faut donc utiliser une table différente pour la sélection et la validation, afin d'éviter de la corrélation lors de la validation.

Deuxième partie

Apprentissage non supervisé

Chapitre 10

Apprentissage non supervisé

Le but de l'apprentissage non supervisé est de trouver des régularités dans les données sans se soucier de la relation entrée-sortie. On recherche ainsi les groupes de variables ou d'objets intéressants et des dépendances entre les variables.

Il existe trois grandes familles de problèmes :

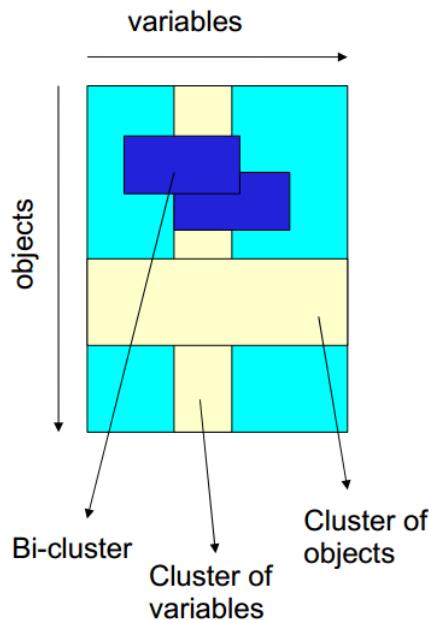
- clustering : trouver des groupes d'échantillons ou de variables.
- réduction de dimensionnalité : on projette les données d'un espace à haute dimension vers un espace plus petit.
- estimation de densité : déterminer la distribution des données dans l'espace d'entrée.

10.1 Clustering

Le but est de grouper une collection d'objets en sous-ensembles (appelés clusters), de façon à ce que chaque objet dans un cluster soit proche des autres, tout en étant éloigné des objets des autres clusters.

Ces groupements peuvent être

- des groupements de lignes/d'objets similaires
- des groupements de colonnes/variables
- du bi-clustering, c'est-à-dire en se basant sur les lignes et les colonnes.



Applications :

- marketing : trouver des groupes de clients qui ont un comportement similaire, en se basant sur leurs caractéristiques et les achats précédents
- biologie : classifier la faune et la flore selon leurs caractéristiques
- web : classification de documents (par exemple des articles de blog)

Deux composantes distinctes sont considérées :

- la mesure de distance entre deux objets
- un algorithme de clustering, qui va minimiser les distances entre les objets d'un groupe et/ou maximiser les distances entre des groupes

10.1.1 Mesure de distances

Distance Euclidienne

Elle mesure la différence entre des coordonnées et pénalise les grosses différences. Il s'agit de la racine carrée de la somme des carrés des différences entre les coordonnées :

$$d_e(x_1, x_2) = \sqrt{(x_{1,0} - x_{2,0})^2 + (x_{1,1} - x_{2,1})^2 + \dots}$$

Distance de Manhattan

Elle mesure la différence entre des coordonnées, mais de manière robuste. Il s'agit de la somme des différences absolues de toutes les coordonnées :

$$d_e(x_1, x_2) = |x_{1,0} - x_{2,0}| + |x_{1,1} - x_{2,1}| + \dots$$

Corrélation

Elle mesure une différence en tenant compte des tendances. La distance entre deux vecteurs est $(1 - \rho)$, où ρ est la corrélation de Pearson entre les deux vecteurs :

$$\rho(x_1, x_2) = \frac{\text{cov}(x_1, x_2)}{\sigma_{x_1} \sigma_{x_2}} = \frac{\sum_{i=1}^n (x_{1,i} - \bar{x}_1)(x_{2,i} - \bar{x}_2)}{\sqrt{\sum_{i=1}^n (x_{1,i} - \bar{x}_1)^2} \sqrt{\sum_{i=1}^n (x_{2,i} - \bar{x}_2)^2}}$$

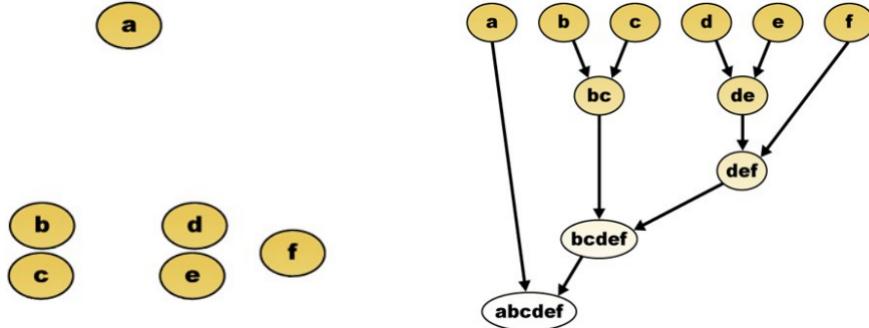
On a que $\rho \in [-1, 1]$, donc $(1 - \rho) \in [0, 2]$: 0 signifie que les données sont fortement corrélées.

10.1.2 Clustering hiérarchique

Algorithme

On a l'algorithme suivant :

1. Chaque objet est assigné à son propre cluster
2. Itérativement :
 - les deux clusters les plus similaires sont joints et rassemblés en un.
 - la matrice de distances est mise à jour avec le nouveau cluster qui en remplace deux.



Distance entre deux clusters

On a plusieurs possibilités :

- **Single linkage** : utiliser la plus petite distance entre deux objets du cluster. Cela a tendance à créer des clusters étalés.

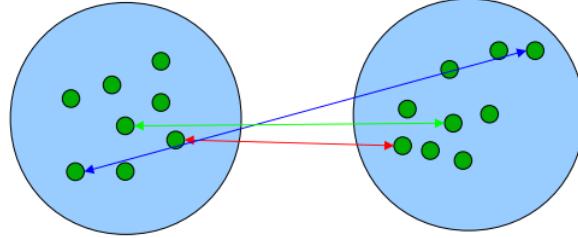
$$d_S(G, H) = \min_{i \in G, j \in H} d_{ij}$$

- **Complete linkage** : utiliser la plus grande distance entre deux objets du cluster. Cela a tendance à créer des grappes.

$$d_C(G, H) = \max_{i \in G, j \in H} d_{ij}$$

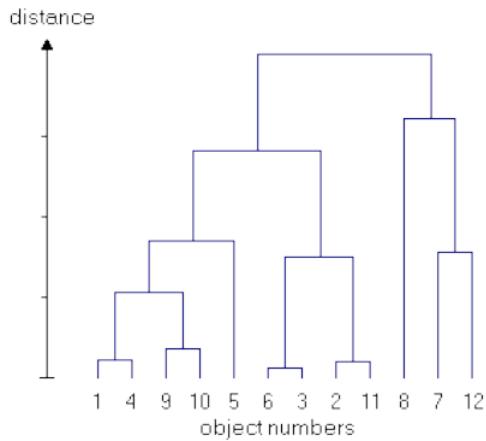
- **Average distance** : calculer la distance moyenne. On obtient un mix des deux autres mesures ; on a une sorte de distance entre les centres de masse.

$$d_A(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d_{ij}$$



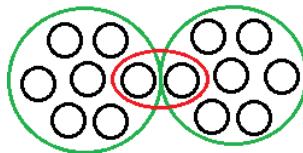
Dendrogramme

Cela permet de visualiser le clustering hiérarchique et de déterminer visuellement le nombre de clusters. Les clusters qui sont unis sont combinés par une ligne. La hauteur d'une ligne est la distance entre les clusters.



Forces et faiblesses

- + on n'a pas besoin de supposer un nombre particulier de cluster
- + on peut utiliser n'importe quel type de matrice de distance
- + on a parfois une interprétation facile des résultats
- trouver une interprétation n'est pas toujours aisé
- une fois qu'il a été décidé de combiner deux clusters, on ne peut pas revenir en arrière. Par exemple, le cluster rouge montre un mauvais départ, alors qu'on aurait voulu obtenir les deux clusters verts :



- pas très bien motivé théoriquement

Algorithme de clustering combinatoire

Soit un nombre de clusters $K < N$ et un encodeur C qui assigne la i ème observation au cluster $C(i)$. On va chercher la fonction C^* qui minimise une fonction de perte, qui mesure si l'objectif de clustering est atteint.

Par exemple, on pourrait avoir comme fonction de perte une qui se base sur l'éparpillement des objets d'un cluster (within cluster scatter) :

$$W(C) = \frac{1}{2} \sum_{k=1}^K \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

Plus les points d'un cluster sont rapprochés, plus ce nombre est petit et donc plus les clusters sont meilleurs. Le nombre de possibilité est cependant trop grand pour une énumération.

$$S(N, K) = \frac{1}{K!} \sum_{k=1}^K (-1)^{K-k} \binom{K}{k} k^N$$

10.1.3 K-means

Cet algorithme effectue un partitionnement avec un nombre k fixé de clusters. On utilise la distance euclidienne entre deux objets, et on va chercher à minimiser la somme des variances intra-cluster :

$$W(C) = \frac{1}{2} \sum_{k=1}^K \sum_{C(i)=k} \sum_{C(i')=k} \|x_i - x_{i'}\|^2$$

Le fait d'utiliser une distance euclidienne permet de réécrire la fonction et d'être plus efficace en passant d'un calcul quadratique par rapport au nombre de cluster à un calcul linéaire sur les points :

$$W(C) = \sum_{k=1}^K N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2$$

avec $\bar{x}_k = (\bar{x}_{1k}, \dots, \bar{x}_{pk})$ le centre du cluster k et N_k le nombre de points dans le cluster k :

$$N_k = \sum_{i=1}^N I(C(i) = k)$$

Cela revient donc à un problème d'optimisation :

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^K N_k \sum_{C(i)=k} \|x_i - m_k\|^2$$

On a deux degrés de liberté : les clusters des points et les centres de masse m_k .

Si C est fixé (donc si on sait à quel cluster appartient chaque point), les m_k sont triviaux à calculer. Si les centres de masse sont fixés, on va chercher les plus proches voisins :

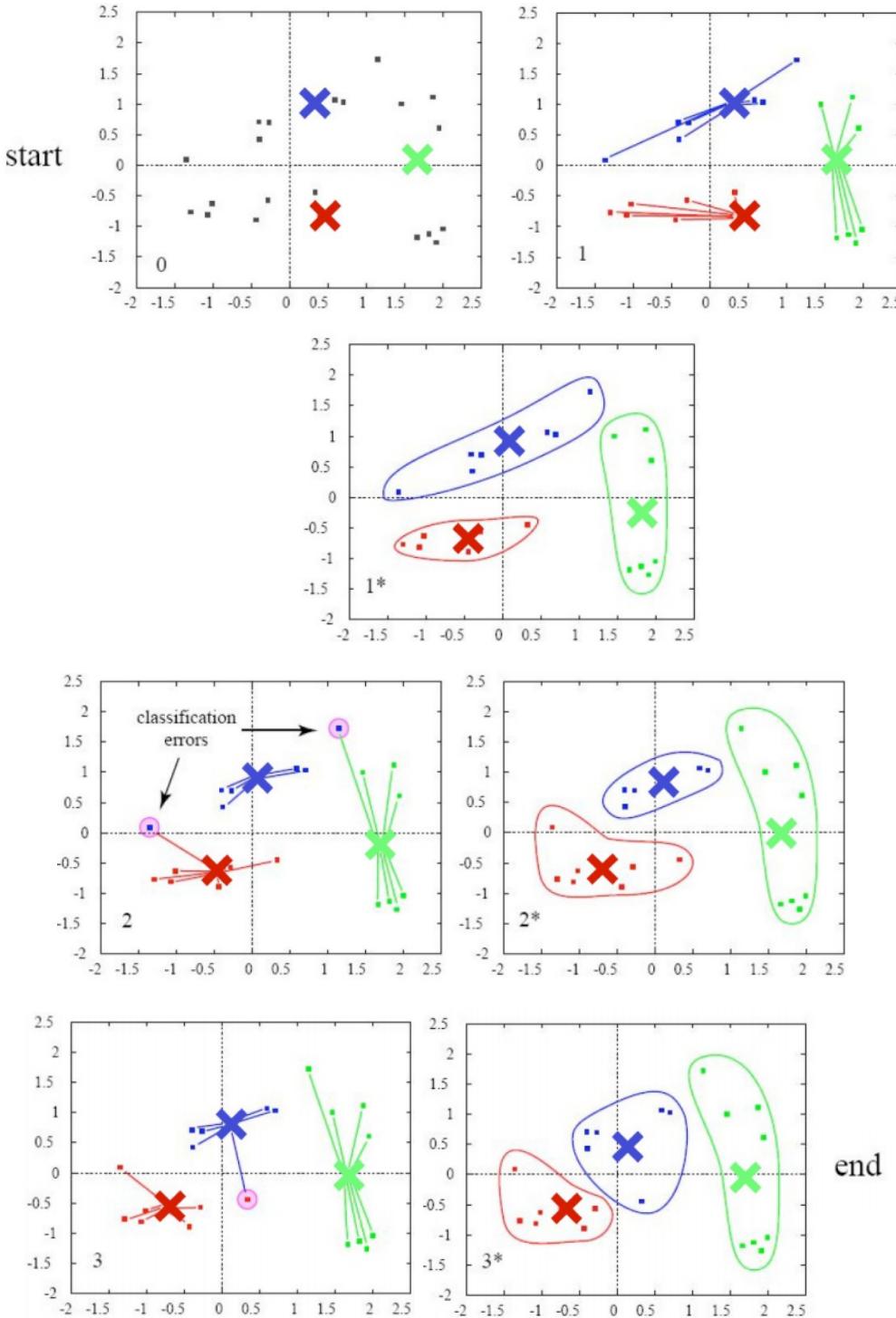
$$C(i) = \arg \max_k \|x_i - m_k\|^2$$

Algorithme

1. On assigne aléatoirement chaque point à un cluster OU on répartit les centres de masse aléatoirement
2. Itérativement :
 - calculer les moyennes des clusters $\{m_1, \dots, m_K\}$
 - pour ces moyennes, assigner chaque observation à la moyenne de cluster la plus proche :

$$C(i) = \arg \min_{1 \leq k \leq K} \|x_i - m_k\|^2$$

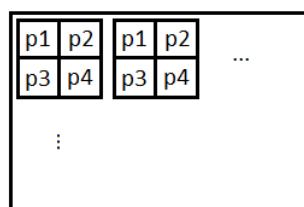
On s'arrête lorsqu'il n'y a plus de changement ; à chaque itération, la somme va diminuer.



Chaque étape réduit l'éparpillement dans les clusters, donc la convergence est assurée, mais uniquement vers un optimum local. De plus, chacune de ces étapes est linéaire par rapport au nombre d'objets, alors qu'avec le clustering hiérarchique il fallait considérer toutes les paires d'objets.

Vu le départ aléatoire de l'algorithme, on pourrait avoir plusieurs solutions différentes. La solution est de redémarrer l'algorithme plusieurs fois.

L'algorithme des k-means peut être utilisé pour de la compression, en découplant une image en blocs et l'appliquant dessus.



p_1	p_2	p_3	p_4	C	m_k
0	255	128	255	1	m_1
\vdots	\vdots	\vdots	\vdots	2	m_2
				1	\vdots
				3	
				4	

K-medoids

Il s'agit d'une extension des k-means qui permet d'utiliser n'importe quelle mesure de distance.

Algorithm 14.2 K-medoids Clustering.

1. For a given cluster assignment C find the observation in the cluster minimizing total distance to other points in that cluster:

$$i_k^* = \operatorname{argmin}_{\{i: C(i)=k\}} \sum_{C(i')=k} D(x_i, x_{i'}). \quad (14.35)$$

Then $m_k = x_{i_k^*}$, $k = 1, 2, \dots, K$ are the current estimates of the cluster centers.

2. Given a current set of cluster centers $\{m_1, \dots, m_K\}$, minimize the total error by assigning each observation to the closest (current) cluster center:

$$C(i) = \operatorname{argmin}_{1 \leq k \leq K} D(x_i, m_k). \quad (14.36)$$

3. Iterate steps 1 and 2 until the assignments do not change.
-

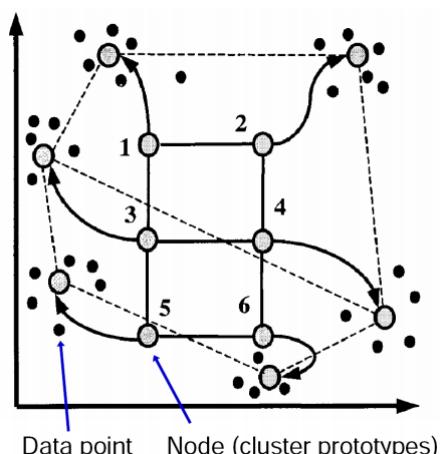
Les k-medoids sont par contre beaucoup plus lents : dans le calcul de l'expression, pour chaque cluster, on doit vérifier toutes les paires de point, ce qui donne un algorithme quadratique.

Forces et faiblesses

- + simple et facile à comprendre
- + on peut clusteriser n'importe quel nouveau point (contrairement au clustering hiérarchique)
- + bonne motivation théorique
- il faut fixer le nombre de clusters
- sensible au choix initial des centres des clusters
- sensible aux données isolées (outliers)

10.1.4 Self-Organizing Maps

C'est une méthode similaire au k-means, mais avec des contraintes supplémentaires : les clusters sont donnés sous forme de matrice (à une ou deux dimensions).



Algorithme

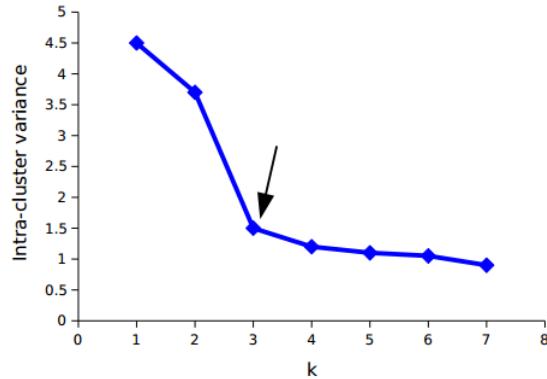
Itérativement :

- prendre P données aléatoirement
- déplacer tous les noeuds dans la direction de P : plus un noeud est dans la topologie mieux c'est (et inversement)
- diminuer la quantité de mouvement autorisée

10.1.5 Nombre de cluster

La question est de savoir quand s'arrêter dans le clustering hiérarchique et comment choisir k pour les k-means et les SOMs. On a un phénomène d'overfitting, comme en apprentissage supervisé :

- on sur-apprend les données s'il y a trop de clusters. On arrive ainsi à trouver des clusters qui n'existent pas dans les données à cause du bruit
 - on sous-apprend les données s'il y a trop peu de clusters, on passe ainsi à côté de clusters pertinents
- Il n'est cependant pas possible, contrairement à l'apprentissage supervisé, d'effectuer une validation croisée. Un choix de nombre de cluster consiste à prendre le point d'inflexion de la courbe de variance intra-cluster.



Autres possibilités :

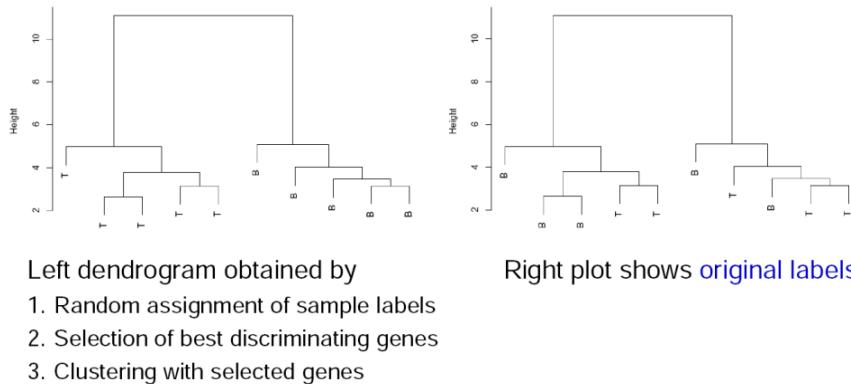
- utiliser des indices internes : sélectionner k qui minimise/maximise les distances intra- et extra-cluster
- gap statistic : utiliser une méthode qui compare un indice interne avec ce qu'on aurait obtenu avec des données aléatoire, et choisir le k qui minimise cette différence
- stabilité : sélectionner le k qui conduit à des clusters stables (calculés avec une analyse bootstrap)

10.1.6 Sélection de features

La sélection de composantes peut améliorer le clustering en diminuant le bruit (et le temps de calcul).

Par exemple, on va garder les 100 variables avec le plus de variance.

Du clustering après une sélection de features supervisée doit être évitée, car on retrouvera toujours la classification, vu que c'est le critère utilisé pour sélectionner les variables.



10.2 Réduction de dimensionnalité

On va chercher à diminuer la dimensionnalité de l'espace des données. On a plusieurs possibilités :

- une sélection de features : on trouve un sous-ensemble des variables originales : $X'_i = X_j$ pour certains j

- une extraction de features : on transforme l'espace original en un espace de plus petite dimension : $X'_i = f(X_1, \dots, X_p)$

- des méthodes linéaires : $f(X_1, \dots, X_p) = w_0 + w_1X_1 + \dots + w_pX_p$

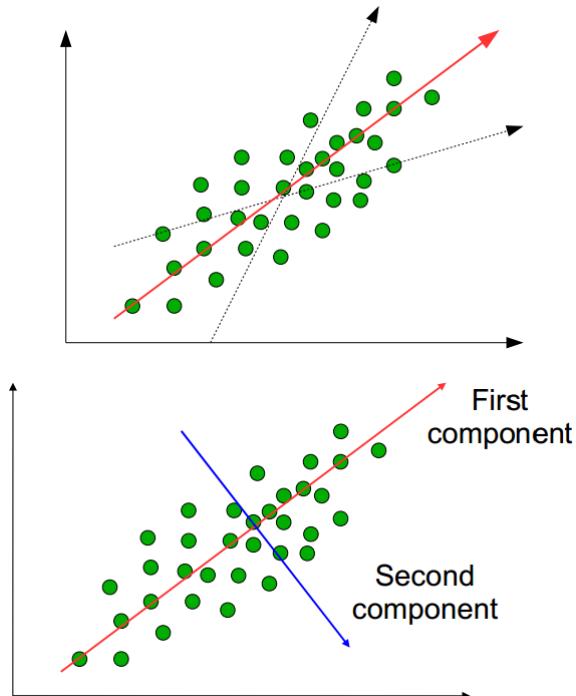
Dans tous les cas, on doit pouvoir reconstruire la base de données d'origine à partir de la version compacte. Les objectifs de la réduction de dimensionnalité sont de

- réduire la dimensionnalité avant de faire passer les données dans d'autres méthodes
- choisir les variables les plus utiles (qui apportent le plus d'information)
- compresser les données
- visualiser des données multidimensionnelles, pour identifier des groupes d'objets et identifier des *outliers*.

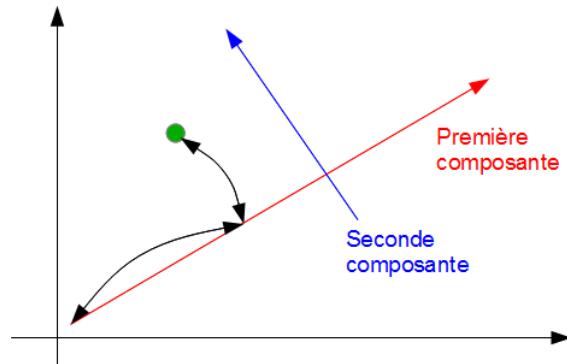
10.2.1 PCA

L'analyse en composantes principales (PCA) est une méthode linéaire qui transforme un grand nombre de variables en un petit ensemble de variables non corrélées que l'on appelle composantes principales. L'idée est de mapper les points dans de petites dimensions, tout en préservant au maximum la variances des données.

On va chercher une direction (composante principale) afin, après projection des données, de maximiser l'étalement. On peut prendre une deuxième composante, afin de conserver plus d'informations, au cas où on aurait trop d'approximations. Chaque point est ainsi codé par sa distance par rapport à une origine.



Cette méthode est très efficace quand il y a beaucoup de corrélation entre les variables (donc de la redondance).



10.2.2 Approche mathématique

On a deux formulations du problème possibles :

- maximisation de la variance : on trouve les directions qui maximisent la variance des données projetées.
- minimisation de l'erreur : on minimise l'erreur de reconstruction des données projetées.

Considérons un ensemble d'observations $\{x_n\}$, $n = 1, \dots, N$, avec x_n un vecteur de dimension D . On veut trouver la direction unitaire u_1 qui maximise la variance de la projection :

$$\arg \max_{u_1} \frac{1}{N} \sum_{n=1}^N \|u_1^T x_n - u_1^T \bar{x}\|^2 = u_1^T C u_1$$

avec

$$\|u_1\| = u_1^T u_1 = 1$$

$$C = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})(x_i - \bar{x})^T$$

$\|u_1^T x_n - u_1^T \bar{x}\|^2$ est la variance du vecteur considéré, car on a

$$\left(\sum_i u_1^T x_i - \underbrace{\frac{1}{N} \sum_i u_1^T x_i}_{\bar{x}} \right)^2$$

$$- u_1^T \sum_i \underbrace{\frac{1}{N} u_1^T x_i}_{\bar{x}}$$

La contrainte de vecteur unitaire ($\|u_1\| = 1$) est nécessaire, sinon le vecteur aurait une variance infinie. Si on introduit le lagrangien :

$$u_1^T C u_1 + \lambda_1 (1 - u_1^T u_1)$$

$$\frac{\partial}{\partial u_1} = 0 \Leftrightarrow C u_1 - \lambda u_1 = 0 \Leftrightarrow C u_1 = \lambda u_1$$

u_1 est un vecteur propre de C . Si on multiplie les deux membres à gauche par u_1^T , on a

$$u_1^T C u_1 = \lambda u_1^T u_1 = \lambda \|u_1\| = \lambda_1$$

On a donc que u_1 est un vecteur propre correspondant à la plus grande valeur propre λ_1 .

Rechercher la seconde composante est similaire, mais on ajoute une contrainte d'orthogonalité. Si on généralise à M composantes, on a que la $M+1$ ème composante est obtenue en maximisant

$$u_{M+1}^T C u_{M+1}$$

avec les contraintes

$$u_{M+1}^T u_{M+1} = 1$$

$$u_{M+1} u_i = 0, \forall i = 1, \dots, M+1$$

Avec le multiplicateur lagrangien :

$$u_{M+1}^T C u_{M+1} + \lambda_{M+1} (1 - u_{M+1}^T u_{M+1}) + \sum_{i=1}^M \eta_i u_{M+1}^T u_i$$

On a l'optimum

$$0 = 2C u_{M+1} - 2\lambda_{M+1} u_{M+1} + \sum_{i=1}^M \eta_i u_i$$

Si on multiplie à gauche par u_i^T , on a

$$0 = 2u_i^T C u_{M+1} - 2\lambda_{M+1} \underbrace{u_i^T u_{M+1}}_0 + \sum_{i=1}^M \eta_i u_i^T u_i$$

Or,

$$u_i^T C u_{M+1} = (C^T u_i)^T u_{M+1} = (C u_i)^T u_{M+1} = (\lambda_i u_i)^T u_{M+1} = \lambda_i u_i^T u_{M+1} = 0$$

On a donc que $\eta_i = 0$ et donc

$$Cu_{M+1} = \lambda_{M+1} u_{M+1}$$

u_{M+1}^T est le vecteur propre de la $M + 1$ ème plus grande valeur propre.

La i ème composante principale pour des objets x_k est donnée par $x'_{ji} = u_i^T x_j$. La reconstruction de l'entrée se fait par

$$\underline{x}_j = \sum_{i=1}^M x'_{ji} u_i = \sum_{i=1}^M (u_i^T \underline{x}_j) u_i$$

PCA minimise également l'erreur de reconstruction :

$$\arg \max_{u_1, \dots, u_M} \frac{1}{N} \sum_{i=1}^N \|x_j - \hat{x}_j\|^2$$

Algorithm 1

Recover basis: Calculate $XX^\top = \sum_{i=1}^t x_i x_i^\top$ and let U = eigenvectors of XX^\top corresponding to the top d eigenvalues.

Encode training data: $Y = U^\top X$ where Y is a $d \times t$ matrix of encodings of the original data.

Reconstruct training data: $\hat{X} = UY = UU^\top X$.

Encode test example: $y = U^\top x$ where y is a d -dimensional encoding of x .

Reconstruct test example: $\hat{x} = Uy = UU^\top x$.

Au final, on obtient un tableau de la forme

	1	2	...	k
\underline{x}_1	$u_1^T \underline{x}_1$	$u_2^T \underline{x}_1$...	$u_k^T \underline{x}_1$
\underline{x}_2				
\vdots				

Les u_k sont les vecteurs propres de la matrice de covariance, les valeurs propres sont quant à elles les variances de la projection.

10.2.3 Etude des composantes

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	PC1	PC2
-0.39	-0.38	0.29	0.65	0.15	0.73	-0.57	0.91	-0.89	-0.17	0.62	-0.33
-2.3	-1.2	-4.5	-0.15	0.86	-0.85	0.43	-0.19	-0.83	-0.4	-2.3	-1.2
0.9	0.4	-0.11	0.62	0.94	0.97	0.1	-0.41	0.01	0.1	0.88	0.31
-0.82	-0.31	0.14	0.22	-0.49	-0.76	0.27	0	-0.43	-0.81	-0.18	-0.05
0.71	0.39	-0.09	0.26	-0.46	-0.05	0.46	0.39	-0.01	0.64	-0.39	-0.01
-0.25	0.27	-0.81	-0.42	0.62	0.54	-0.67	-0.15	-0.46	0.69	-0.61	0.53

Scores for each sample and PC

$$PC1 = 0.2 * A1 + 3.4 * A2 - 4.5 * A3$$

$$VAR(PC1) = 4.5 \rightarrow 45\%$$

$$PC2 = 0.4 * A4 + 5.6 * A5 + 2.3 * A7$$

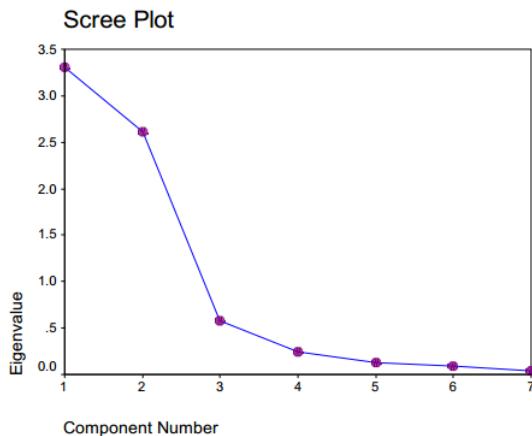
$$VAR(PC2) = 3.3 \rightarrow 33\%$$

...

...

Le poids de chaque variable donne une idée de son importance dans une composante, et peut être utilisé pour de la sélection de features. Pour chaque composante, on a une mesure du pourcentage de la variance des données initiales qu'elles contiennent.

La question est de savoir combien de composantes il faut prendre. Pour cela, on trace le scree plot, les valeurs propres (\Leftrightarrow variances) de chaque composante en ordre décroissant.



On supprime les composantes avec des valeurs propres inférieures à 1 et on prend k au point d'infexion de la courbe, à partir duquel les débris commencent à s'accumuler.

10.2.4 Limitations

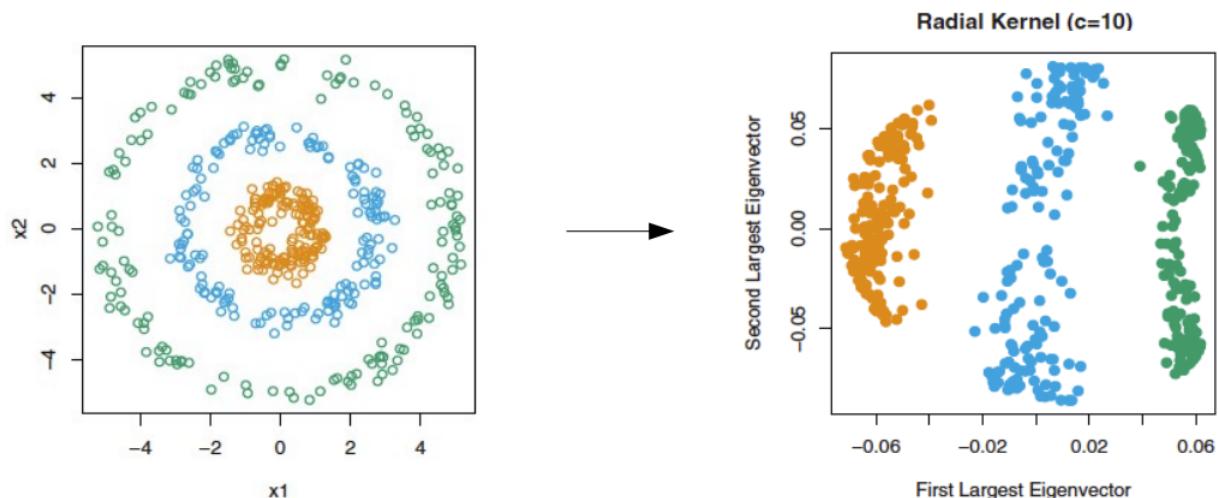
PCA peut retrouver visuellement des groupes de données, mais s'il échoue à retrouver ces groupes, on ne peut rien conclure, et c'est indirect par rapport aux méthodes de clustering.

PCA peut être utilisé pour de la sélection de composantes, mais les premières composantes ne sont pas forcément liées à la sortie et les méthodes de sélection de features supervisées fonctionnent mieux. De ce fait, PCA ne doit être considéré que comme un outil d'exploration.

10.2.5 Extensions de PCA

Kernel PCA

Il s'agit d'une technique d'extraction de features basée sur le noyautage de PCA.



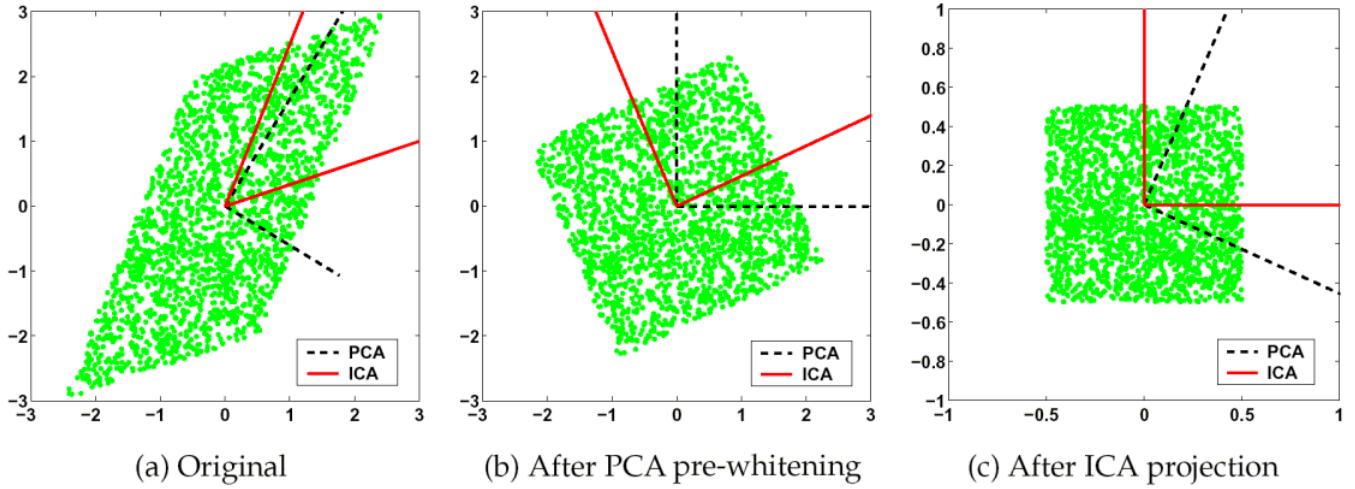
Sparse PCA

On cherche les composantes avec des données creuses, avec peu de composantes avec des poids non nuls. On utilise des pénalisations différentes, comme par exemple L1 (LASSO).

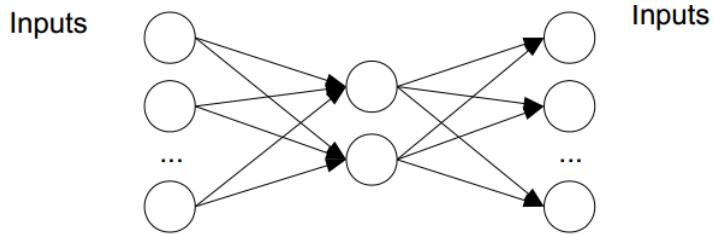
10.2.6 Autres techniques de réduction de dimension

Independent Component Analysis (ICA)

On ne se force pas à utiliser des composantes orthogonales.



Auto-encodeur avec des réseaux de neurones



Multi-dimensional scaling (MDS)

On cherche des nouvelles coordonnées telles que certaines distances sont respectées (au sens des moindres carrés).

Find $z_1, z_2, \dots, z_N \in \mathbb{R}^k$ that minimize:

$$S_M(z_1, z_2, \dots, z_N) = \sum_{i \neq i'} (d_{ii'} - \|z_i - z_{i'}\|)^2$$

10.3 Autres méthodes non-supervisées

- règles associatives
- estimation de densité :

 - modèles de mixture
 - réseau bayesiens