Apprentissage sur un petit ensemble de données

Bousquet Jérémie, Hmamouche Youssef

Section 1

Apprentissage et sur-apprentissage

Apprentissage automatique - problématique

Algorithmes permettant de résoudre un problème, une tâche, en apprenant des données - lorsqu'il n'est pas possible ou complexe d'écrire un algorithme pour résoudre la tâche.

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \text{ et } y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}, \text{ on suppose } y = f(X) + \epsilon$$

- Chaque ligne de X est une donnée structurée (un exemple), composée de p variables (features)
- A chaque exemple est associée une cible y_i, ce que l'on souhaite prédire
- $y = f(X) + \epsilon$ est ce que l'algorithme cherche à approximer (ϵ: l'erreur)
- Si les y_i sont continus on parle de tâche de régression, s'ils sont discrets on parle de tâche de classification.

Apprentissage automatique - problématique

Classification: reconnaître un objet dans une image (chaque ligne de X est une image, les p variables représentant les pixels de l'image, et par exemple $y \in \{chien, chat, ours, ...\}$)

Régression: prédiction du prix de vente d'une maison en fonction de critères (les p variables sont les critères, superficie, nombre de chambres, etc, et y est le prix de vente)

Ces algorithmes sont généralement utilisés en deux phases:

- Apprentissage: on approxime $f(X) = y$, X et y étant connus (apprentissage supervisé), $\Rightarrow \widehat{f}$
- Inférence: on prédit y' à partir de $\widehat{f}$ apprise et de nouvelles données X' pour lesquelles y' attendu peut être connu (pour tester l'apprentissage) ou pas (problème réel).

Autres formes d'apprentissage : non supervisé (pas de y, ex. clustering), partiellement supervisé, multi-label (y_i a n dimensions), ...

Apprentissage automatique - dilemme biais-variance

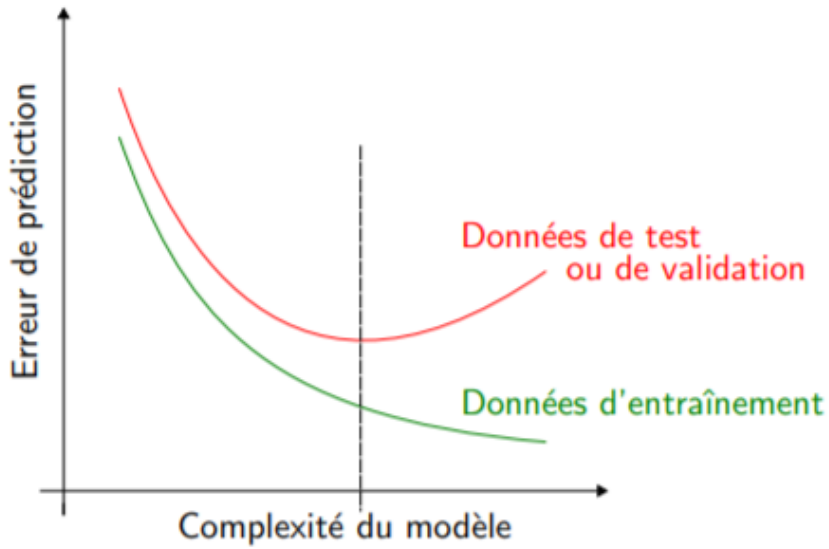
- Biais inductif: on commence par faire une hypothèse dans le choix d'une famille de fonctions $\mathbb{H}$ pour approximer f (ex. pour la régression: polynomiale, logistique..., pour la classification: Support Vector Machines, Bayésien, réseau de neurones ...).
- L'erreur commise sur plusieurs jeux de données peut se décomposer en biais, variance, et une erreur irréductible liée au bruit dans les données du problème.

$$E[(y - \hat{f}(x))^2] = Biais[\hat{f}(x)^2] + Var[\hat{f}(x)] + \sigma^2$$

- Biais : écart entre la fonction de prédiction moyenne $\hat{f}$ apprise sur plusieurs jeux de données, et la fonction recherchée f,
- Variance : écart entre la fonction de prédiction moyenne apprise sur plusieurs jeux de données, et une fonction de prédiction apprise sur un seul jeu de données.

Apprentissage automatique - dilemme biais-variance

- Si la complexité du modèle $\mathbb{H}$ choisi est faible, le biais sera important et la variance faible
- Si la complexité du modèle $\mathbb{H}$ choisi est forte, le biais sera faible et la variance importante



Si le biais est trop important on parle de sous-apprentissage. Si la variance est trop importante on parle de sur-apprentissage (overfitting - une variation même faible des données, peut engendrer une réponse très différente de l'algorithme). (voir aussi ce cours)

Overfitting - comment le mesurer ?

- Evaluer la différence entre les erreurs des modèles sur les données d'apprentissage, de validation, et de test. Pour la régression, on peut appliquer des tests statistiques, pour la classification il est préférable d'évaluer la variance des performances mesurées (en effectuant plusieurs mesures).

Généralement on découpe les données en trois ensembles distincts : données d'apprentissage (le modèle ajuste ses paramètres internes sur ces données), données de validation (pour choisir les meilleurs hyper-paramètres d'un modèle, on compare les performances obtenues sur cet ensemble suivant les hyper-paramètres utilisés), et données de test (pour évaluer la capacité de généralisation du modèle sur des données totalement nouvelles pour lui).

Overfitting - comment le mesurer ?

- Par conséquent, on peut mesurer l'overfitting par le calcul de la différence entre les vecteur des erreurs de prédiction d'entrainement et de test :

$$E_{training} - E_{test}$$

- Il y a overfitting is $E_{training} << E_{test}$, et underfitting si $E_{training} >> E_{test}$.
- Il n'y a pas une formule générale pour la quantification de l'overfitting, mais on peut l'évaluer par la variance ou l'écart-type:

$$\text{Variance } (E_{training} - E_{test})$$

Overfitting - comment le mesurer ?

- Eviter les biais méthodologiques (cf. par ex. Cawley et al. (2017))
  Exemples:
  - Si des données sont utilisées pour apprendre **ou sélectionner** un modèle, alors elles ne doivent pas servir à évaluer ce modèle (risque d'overfitting sur l'apprentissage mais aussi sur la sélection du modèle).
  - Etre attentif aux métriques permettant d'évaluer objectivement le modèle (ex. pour la classification, ne pas se limiter à la précision qui pénalise les faux positifs, mais aussi le rappel qui pénalise les faux négatifs, ou le f-score qui combine les deux).
  - Choisir une méthode adaptée pour la découpe en train/valid/test. Par exemple si les classes sont déséquilibrées utiliser une découpe stratifiée (qui respecte le ratio entre classes dans chaque ensemble). Un modèle aléatoire peut avoir une très bonne précision sur des données aux classes fortement déséquilibrées !

Overfitting - comment l'éviter ?

Des méthodes existent, généralement très dépendantes de la famille de modèles choisie ($\mathbb{H}$)

- Contraindre les valeurs d'hyper-paramètres recherchés suivant la connaissance de la famille de modèle.

Par exemple des valeurs extrêmes pour C et γ pour les SVMs peuvent encourager l'overfitting (mais il n'y a pas de règle absolue) - les hyper-paramètres peuvent en fait influer sur la complexité du modèle (et donc le dilemme biais-variance). Les connaissances empiriques et théoriques sur chaque famille de modèle peuvent permettre d'ajuster la recherche d'hyper-paramètres pour éviter sous- et sur-apprentissage.

Overfitting - comment l'éviter ?

- Régulariser l'apprentissage

  Suivant la famille de modèles, la régularisation consiste à ajouter un terme de contrainte dans la fonction de perte, l'objectif étant généralement de pénaliser les modèles trop complexes lors de l'apprentissage. Note: la fonction de perte (ou coût) est une fonction que l'on cherche à minimiser lors de l'apprentissage. Limiter la complexité d'un modèle revient généralement à limiter l'espace de recherche de ses paramètres (on ne parle ici que des modèles paramétriques).

  Par exemple:

  - la régularisation l_2 contraint des paramètres "petits" (proches de zéro)
  - la régularisation l_1 contraint des solutions parcimonieuses (comportant des paramètres nuls)

Overfitting - comment l'éviter ?

Exemple: Régression Ridge:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \boxed{\lambda \sum_{j=1}^{p} \beta_j^2}$$

Les β_j forment les poids que l'algorithme cherche à apprendre en minimisant cette fonction de coût. La présence du terme de pénalisation l_2 contraint l'algorithme à tendre vers des solutions β_j "petites", donc à limiter l'espace de recherche des solutions, donc à limiter la complexité du modèle. L'importance de cette régularisation est conditionnée par le paramètre λ.

Overfitting - comment l'éviter ?

- "Augmenter" les données

On entend ici ajouter de nouvelles données synthétisées à partir des données existantes et par ajout d'un "bruit" suivant différentes méthodes (pour des images, on peut ajouter de nouvelles images en appliquant des transformations de type flip, rotations, bruit gaussien, recadrages, etc).

En augmentant le bruit dans les données, à complexité équivalente il sera plus "difficile" pour le modèle de s'adapter étroitement aux données, la variance aura tendance à être plus faible ainsi que l'overfitting. Ce "bruit" ne doit cependant pas masquer l'information que l'on cherche à apprendre.

Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

- Méthode utilisée : k-fold cross-validation avec ensemble de test et de validation.
- Problèmatique : la cross validation pose quelques problèmes pour les données séquentielles car elle ne tient pas en compte l'ordre chronologiques des données, mais on peut la faire marcher dans notre cas si on considère chaque conversation comme un sous-ensemble sous l'hypothèse que l'ordre des conversations n'est pas important.

Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

- Première stratégie : construire un seul modèle pour toutes les conversations.
- Dans ce cas, on peut découper les données en 4 blocks (comme découpé lors de l'expérience d'IRMf), chaque block contient 6 conversations. Sur chaque block on peut appliquer une k-fold cross-validation en gardant une seule conversation comme données de test. Pour les autres, on change aléatoirement l'ordre des conversations à chaque fois et fixant une conversation pour la validation, et on répète ce processus, jusqu'à ce que chaque conversation des données d'entrainement est utilisée une fois pour la validation.
- Deuxième stratégie : deux modèles séparés, un pour les conversations humain-humain (HH) et l'autre pour les conversations humain-robot (HR).

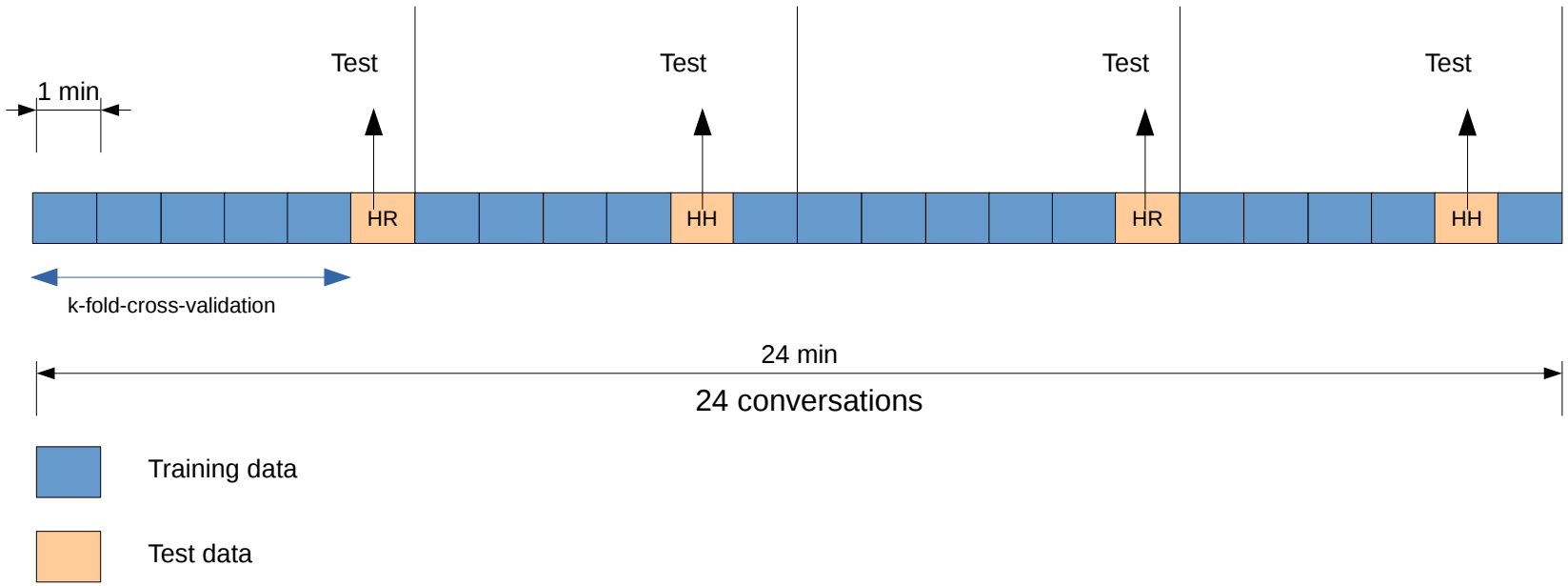
Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

Modélisation 1

- 20 conversations pour l'entrainement, et 4 conversations de test (2 HH et 2 HR).
- 5-fold cross-validation avec ensemble de test et de validation sur les 4 blocks :
  - Division des conversations en 4 blocks.
  - Sur chaque block, une conversation est extraite comme données de test.
  - Application d'une 5-fold cross-validation sur le reste des conversations.

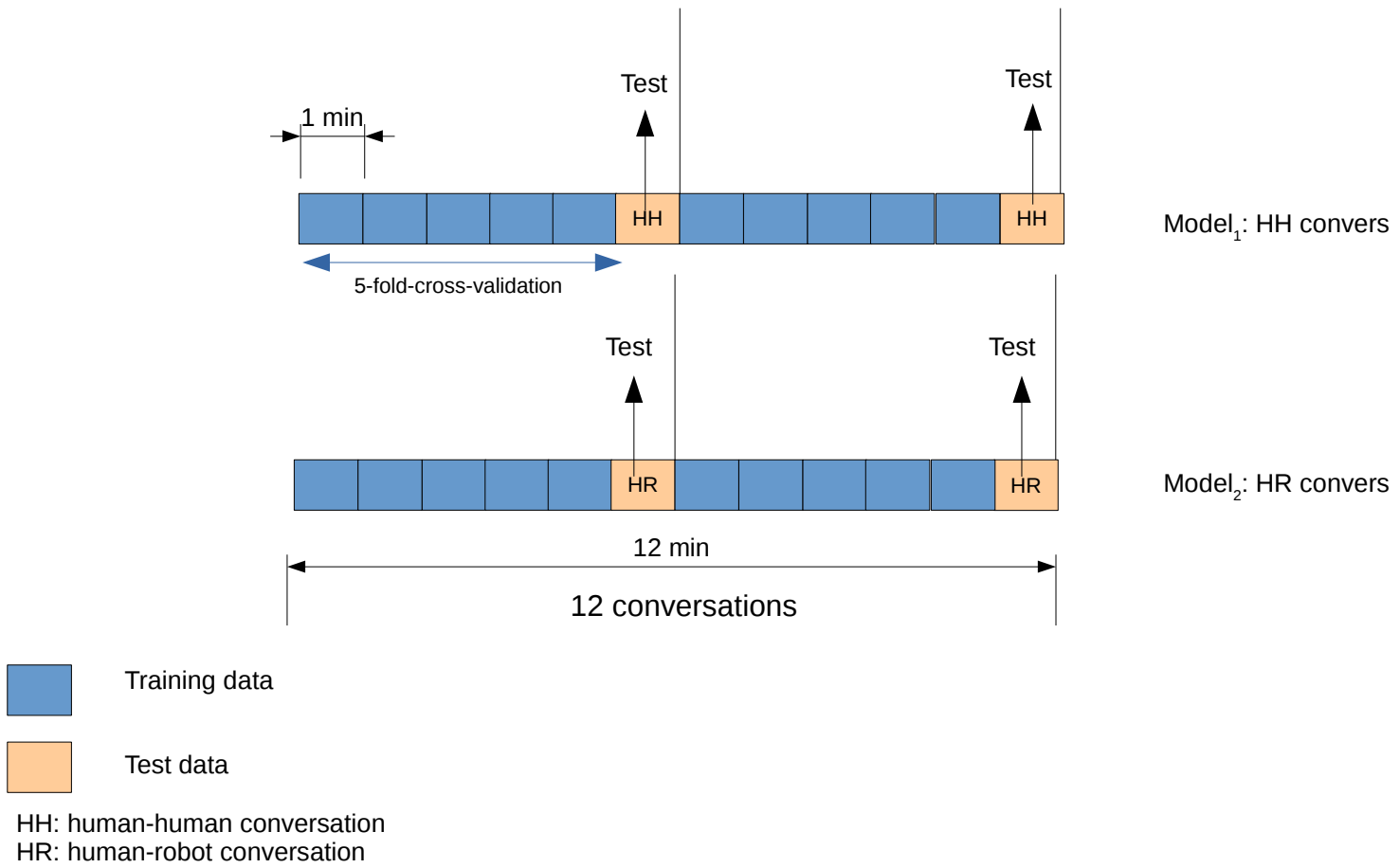
Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

Modélisation 1 : un modèle pour toutes les conversations



Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

Modélisation 2 : deux modèles selon le type des conversations

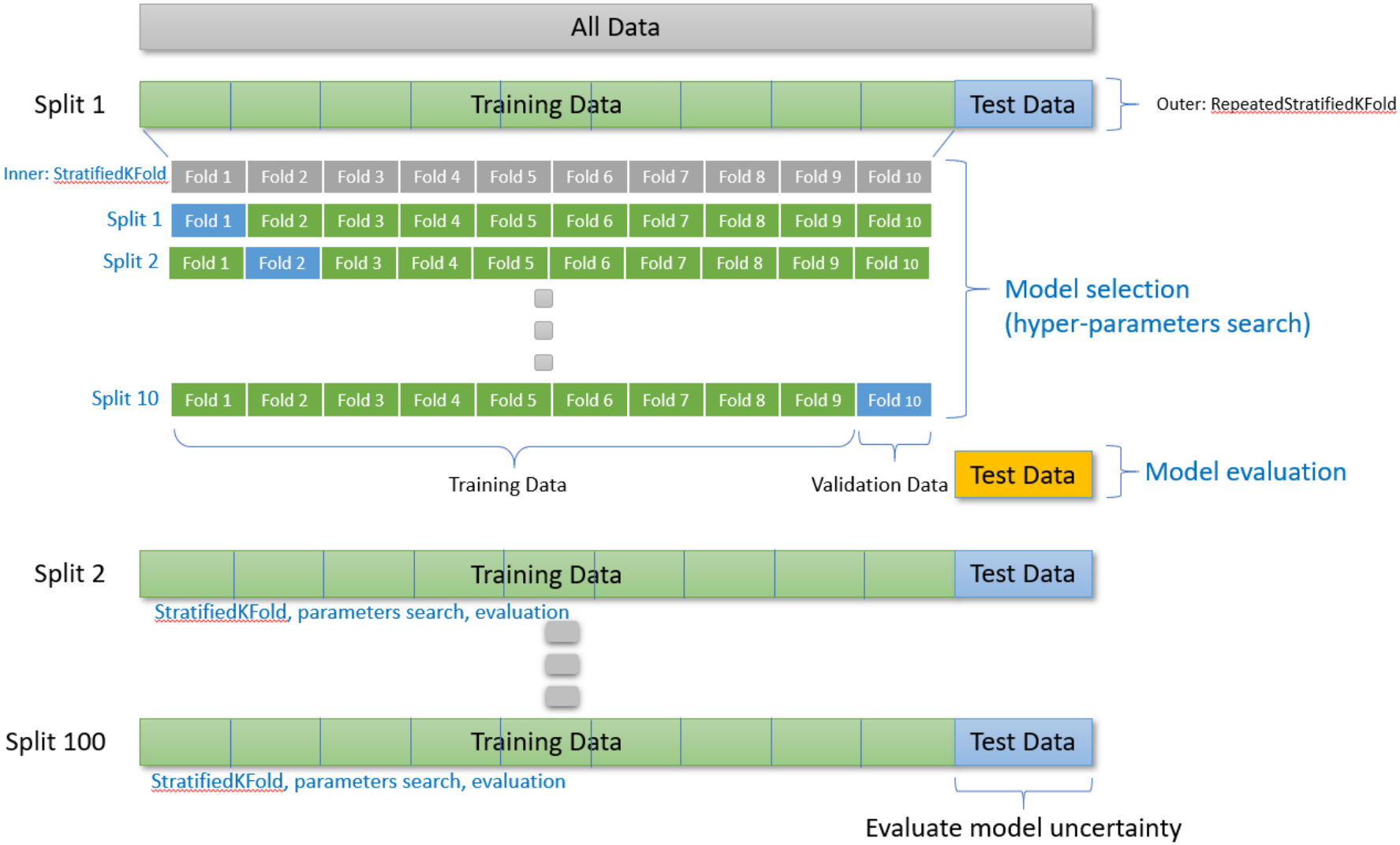


Application - Prédiction du sentiment de (co)présence

Méthodologie:

- discrétisation des scores de présence/co-présence en problèmes de classification binaire
- $n \times$ 10-folds (90% train / 10% test), splits aléatoires stratifiés (= conservant les proportions de chaque classe)
  - 10-fold cross-validation sur l'ensemble train pour la recherche d'hyper-paramètres du modèle (chaque fold sert d'ensemble de validation à son tour)
  - évaluation de la capacité de prédiction sur l'ensemble test (non vu lors de l'apprentissage ou de la sélection de modèle)
  - moyennage des scores de test sur les $n \times 10$ splits (avec calcul de variance et intervalle de confiance)

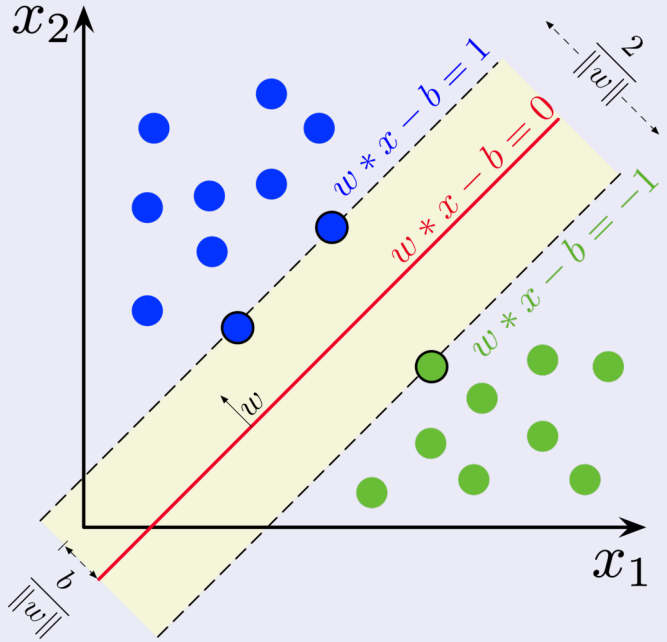
Application - Prédiction du sentiment de (co)présence



Application - Prédiction du sentiment de (co)présence

Overfitting: cas du Support Vector Machines (SVM) (1/4)

Le classifieur SVM peut-être vu comme un problème d'optimisation sous contrainte : maximiser une marge entre hyper-plan et données.



$$\underset{w,b}{\text{minimize}} \quad \frac{1}{2}\|w\|$$

$$\text{subject to} \quad y_i(w^t.x_i + b) \geq 1$$

Application - Prédiction du sentiment de (co)présence

Overfitting: cas du Support Vector Machines (SVM) (2/4)

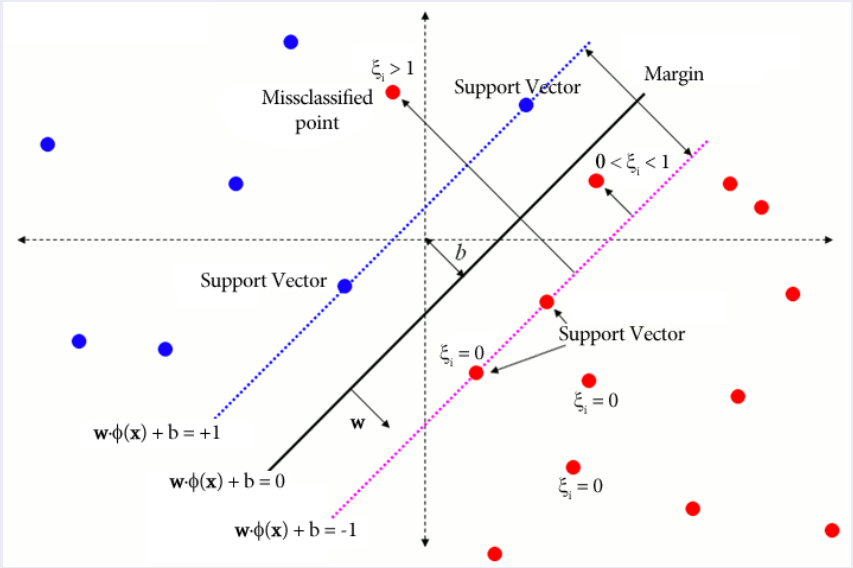
Il n'est pas forcément possible de séparer linéairement les données pour tout problème de classification. Parfois, des exemples d'apprentissage dits "outliers", trop excentriques, peuvent rendre le problème impossible à résoudre. Afin de relâcher la contrainte et autoriser la mauvaise classification de certains exemples, sont introduites les slacks variables ξ_i:

$$\begin{aligned} \underset{w,b}{\text{minimize}} \quad & \frac{1}{2}\|w\| + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w^t.x_i + b) \geq 1 - \xi_i \end{aligned}$$

Le terme à minimiser $C\sum_{i=1}^{n}\xi_i$ limite l'influence des slacks variables et est conditionné par C qui est un des hyper-paramètres du SVM.

Application - Préduction du sentiment de (co)présence

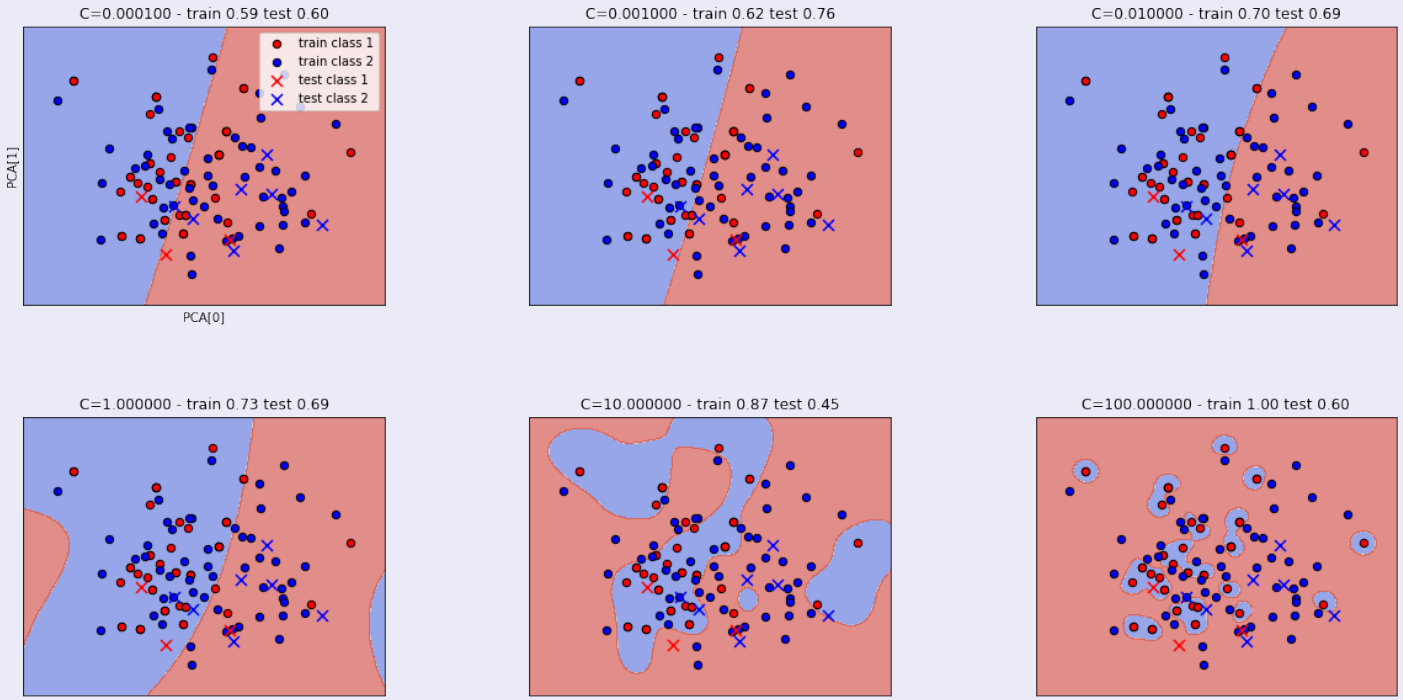
Overfitting: cas du Support Vector Machines (SVM) (3/4)



Plus la valeur de C est grande, moins les points “outliers” sont autorisés, et plus la marge aura tendance à être petite. A l’inverse, plus C est petit, plus on autorise les “outliers” et plus la marge aura tendance à être grande. Il faut donc trouver le meilleur compromis pour obtenir une frontière de séparation et une marge qui permette une bonne classification des exemples d’apprentissage, tout en conservant sa capacité de généralisation pour des points non vus lors de l’apprentissage.

Application - Prédiction du sentiment de (co)présence

Overfitting: cas du Support Vector Machines (SVM) (4/4)



Le score d'apprentissage augmente quand C augmente (on autorise moins d'outliers donc moins d'exemples incorrectement classifiés), mais le score de test ne s'améliore pas au contraire. Le biais du classifieur se réduit, mais logiquement sa variance augmente, et les résultats sur l'ensemble de test peuvent montrer une grande variabilité (par exemple ici le score de test n'est pas minimal pour C=100, bien que l'overfitting soit flagrant).

Section 2

Méthodes et techniques adaptées aux petits ensembles de données

Méthodes et techniques adaptées aux petits ensembles de données

- Utiliser des algorithmes reconnus pour leurs bonnes performances sur les petits ensembles de données (cf. par exemple C.Salperwyck et V.Lemaire 2010, Bergtold et al. 2011)
  - *Random Forests* (ensemble d'arbres de décision entraînés sur des réalisations différentes de l'ensemble de données d'origine, leurs nombre et variations peuvent compenser la variance dûe à un petit ensemble de données)
  - *Bayésien naïf* (probabilités calculées avec le postulat de variables non corrélées)
  - *Régression logistique* (classification binaire)

... mais tout algorithme **peut** convenir (probablement sous réserve de ne pas être trop complexe), seuls des tests peuvent réellement renseigner sur l'algorithme le plus approprié.

Méthodes et techniques adaptées aux petits ensembles de données

- Augmenter les données, pour:
  - obtenir un dataset plus grand
  - et/ou réduire les déséquilibres entre classes
- Diminuer l'influence du bruit / le biais, retirer les outliers
  - retirer les "outliers" manifestement hors de la distribution normale des échantillons (erreurs de mesures flagrantes par exemple) - certains algorithmes (SVM) peuvent aider à la sélection d'échantillons.
  - régulariser l'apprentissage

Sur un faible volume de données le bruit, les outliers, peuvent avoir un impact important. De façon générale, éviter les modèles trop complexes, qui risquent d'overfitter rapidement sur un faible volume de données.

Techniques pour la génération de nouvelles données

- *Random sampling* : re-sampling (tirage avec remise), réalisation d'un nouveau dataset à partir du dataset existant, avec potentiellement des échantillons en moins, et d'autres dupliqués.
  - Avantages: utilisé notamment pour corriger les déséquilibres entre classes (upsampling, downsampling), ou encore pour le bootstrap des random forests.
  - Inconvénients: pas vraiment de génération de nouvelles données, risque d'overfitting sur la classe minoritaire.
- *SMOTE, ADASYN* : synthèse de nouveaux examples par interpolation à partir d'exemples existants.
  - Avantages: réduit le risque d'overfitting du simple random sampling.

Techniques pour la génération de nouvelles données

- Réseaux de neurones dont les réseaux antagonistes génératifs (GAN (Ian Goodfellow et al 2014))

Un réseau apprend à générer des données, et à tromper un autre réseau apprenant à discriminer vraie donnée et donnée générée (apprentissage souvent difficile).

- Avantages: données synthétiques potentiellement plus réalistes.
- Inconvénients: nécessite beaucoup de données d'entrainement à l'heure actuelle donc peu utilisable pour un ensemble de données petit.

Application - Prédiction de l'activité cérébrale en fonction des signaux multimodaux

Génération de nouvelles données

- Il serait intéressant d'ajouter de nouvelles données si cela permet d'améliorer la qualité des prédictions.
- Pour le moment, pour chaque sujet, nous avons 1200 observations, où chaque conversation contient 50 observations.
- Ces observations présentent des auto-corrélations pour la plupart des variables.
- Par conséquent, il faut générer des données de manière à tenir en compte ces auto-corrélations.

Application - Prédiction du sentiment de (co)présence

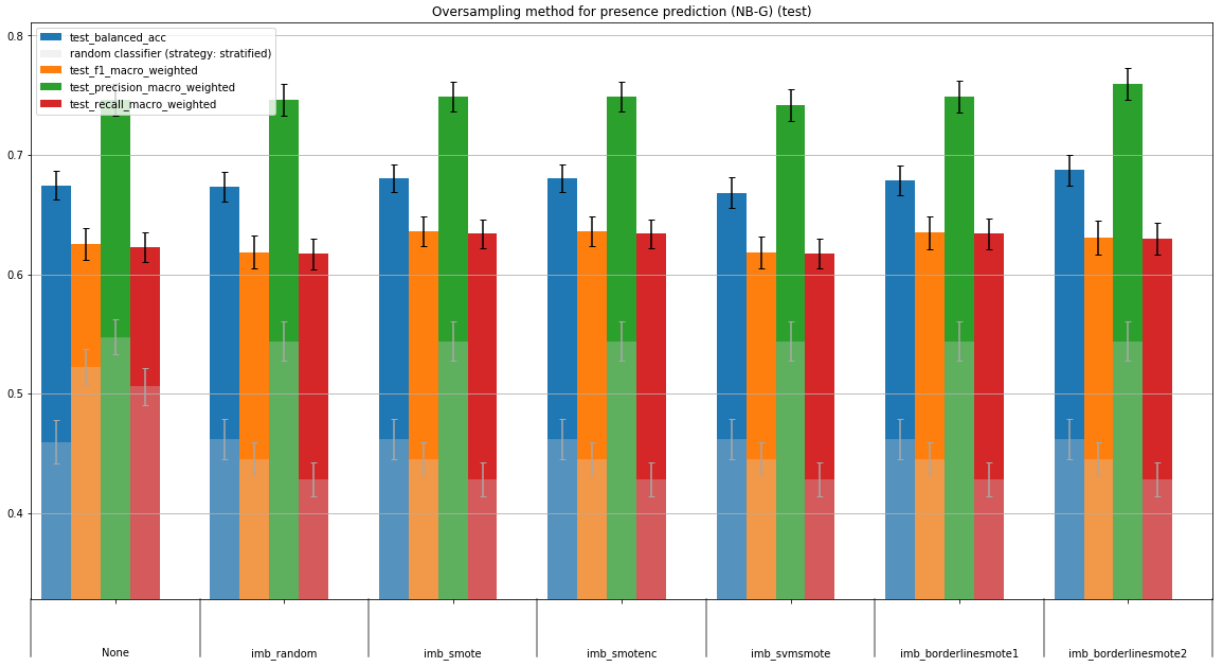
- GAN
  - peut nécessiter de grands volumes de données

    approche “fine-tuning”, mais possible seulement si le domaine/la tâche est semblable, compliqué ici
  - qualité / pertinence des données générées difficile à évaluer

    ... mais impact sur l'apprentissage évaluable

Application - Prédiction du sentiment de (co)présence

- Random Sampling, SMOTE, ADASYN, ...
  - Faciles à mettre en oeuvre, variables continues interpolables
  - Variables catégorielles: méthodes pour déterminer la catégorie de la nouvelle donnée (plus proches voisins ...)
  - expérimentalement pas de réel avantage mesuré du moment que les métriques prennent en compte le déséquilibre de classes:



Application - Prédiction du sentiment de (co)présence

Remarque: un déséquilibre entre classes induit un biais du classifieur, que l'on peut corriger soit en rééquilibrant (sampling/synthèse), soit de façon équivalente en prenant en compte le déséquilibre dans la fonction de coût à optimiser.

En pratique si l'évaluation des performances utilisée pour optimiser les hyper-paramètres, prend en compte la distribution des classes, il semble que ce biais soit (ici du moins) efficacement compensé.