Cours de Data Science

Marc Tommasi

11 octobre 2022

Outline

Outline

Idées informelles

- Utilisé pour expliquer comme pour prédire. Exemple : expliquer la consommation en fonction du poids d'une la voiture.
- On prend la classe des droites $y = ax + b$. On cherche a et b étant donné des poids (x) et la consommation observée (y).
- On appelle aussi b *intercept* (l'ordonnée à l'origine) et a est un coefficient (qui définit la pente de la droite).
- Si on sait que les données x sont centrées en 0 alors l'intercept est nul.
- Généralisation à la dimension n. Exemple : prédire la consommation en fonction du poids de la voiture et de la puissance du moteur, de la météo, de la vitesse, etc.
- On dispose de m exemples de la forme $((x_1, x_2, \ldots, x_d), y)$
- On prend une classe $\mathcal{H}$ de fonctions linéaires : on cherche les w_i tels que $y = w_0 + w_1 x_1 + \cdots + w_d x_d$
- Remarque : on peut poser $x_0 = 1$ et résoudre $y = w_0 x_0 + w_1 x_1 + \cdots + w_d x_d = \boldsymbol{w}^\top \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

ERM

- $\mathcal{X} \subseteq \mathbb{R}^d$, $\mathcal{Y} \subseteq \mathbb{R}$ et $\mathcal{H} = \{\boldsymbol{x} \to \langle \boldsymbol{w}, \boldsymbol{x} \rangle \mid \boldsymbol{w} \in \mathbb{R}^d\}$.
- On utilise la perte quadratique : $\ell(h, (\boldsymbol{x}, y)) = (h(\boldsymbol{x}) - y)^2$

ERM, fonction objectif

- On cherche à minimiser l'erreur quadratique pour chaque exemple et la fonction de perte, fonction d'objectif est

$$\underset{w}{\operatorname{argmin}} \frac{1}{m} \sum_i (\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - y_i)^2$$

(MSE : Mean Square Error, moyenne des erreurs quadratiques)

- Notation matricielle si toutes les lignes de X sont les données et les colonnes sont les attributs

$$\underset{w}{\operatorname{argmin}} \|X\boldsymbol{w} - \boldsymbol{y}\|^2$$

(on a viré le $\frac{1}{m}$)

- l'objectif est convexe : il y a un minimum global
- on peut le calculer analytiquement.

Dérivées et gradient

- La dérivée f' d'une fonction $f : \mathbb{R} \to \mathbb{R}$ évaluée en x est définie par

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

- La dérivée s'annule pour un x qui minimise f.
- Pour $f : \mathbb{R}^d \to \mathbb{R}$, on peut considérer la dérivée par rapport à chaque composante de $\boldsymbol{x} = (x_1, \ldots, x_d)$. C'est noté $\frac{\partial f}{\partial x_i}$.
- Le vecteur de dimension d de toutes ces fonctions dérivées est le gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d})$.
- Rappel : la dérivée de $f(x) = x^n$ est $f'(x) = nx^{n-1}$ et la dérivée d'une constante est nulle ;

Bilan

- On peut trouver une solution analytique qui sera $(X^\top X)^{-1} X^\top \boldsymbol{y}$.
- Informellement, on cherche $X\boldsymbol{w} = \boldsymbol{y}$, qu'on peut réécrire $X^\top X\boldsymbol{w} = X^\top \boldsymbol{y}$, puis $\boldsymbol{w} = (X^\top X)^{-1} X^\top \boldsymbol{y}$.
- Remarque
  - La matrice $(X^\top X)$ peut ne pas être inversible, on peut calculer plutôt $X^\dagger \boldsymbol{y}$ où $X^\dagger$ est la Moore-Penrose pseudo-inverse, qui elle se calcule par une SVD (Si $X = U\Sigma V^\top$ alors $X^\dagger = V\Sigma^\dagger U^\top$ et $\Sigma^\dagger$ est calculée en prenant l'inverse des valeurs sauf pour les 0 qui restent 0).

Complexité

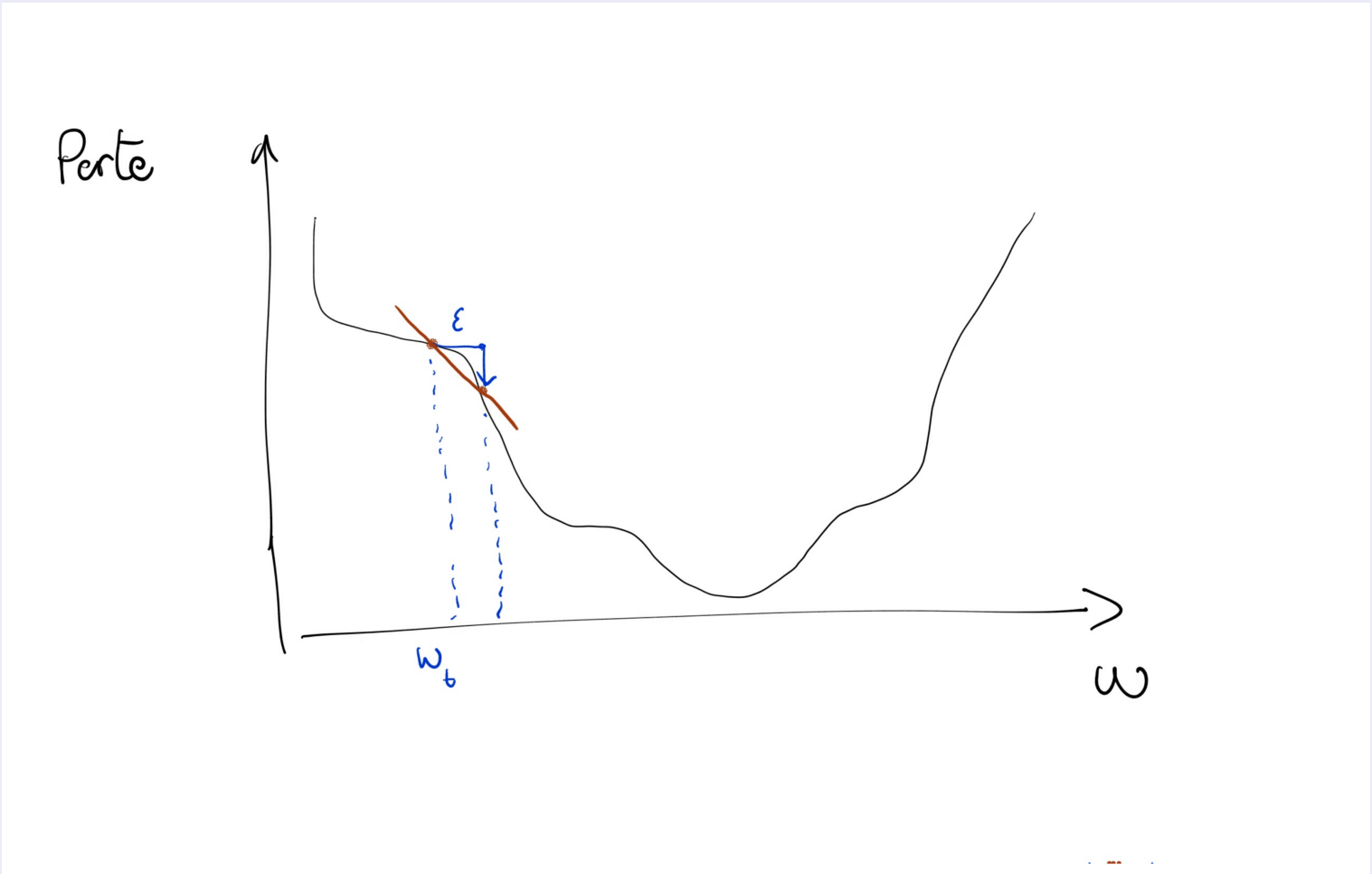
- Le calcul analytique est coûteux pour de très grands jeux de données
- En $O(n^3)$ pour la plupart des algos (voire $O(n^{2.4})$ pour les meilleurs).
- Une approche par descente de gradient permet d'avoir un algorithme quadratique qui marche plutôt bien en pratique.

Descente de gradient I

- On cherche à faire baisser l'erreur empirique pas à pas : trouver la modification des paramètres pour que la fonction de perte prenne une valeur plus petite
- Idée de descente par la plus forte pente sur la courbe de la perte en fonction des paramètres

Descente de gradient II

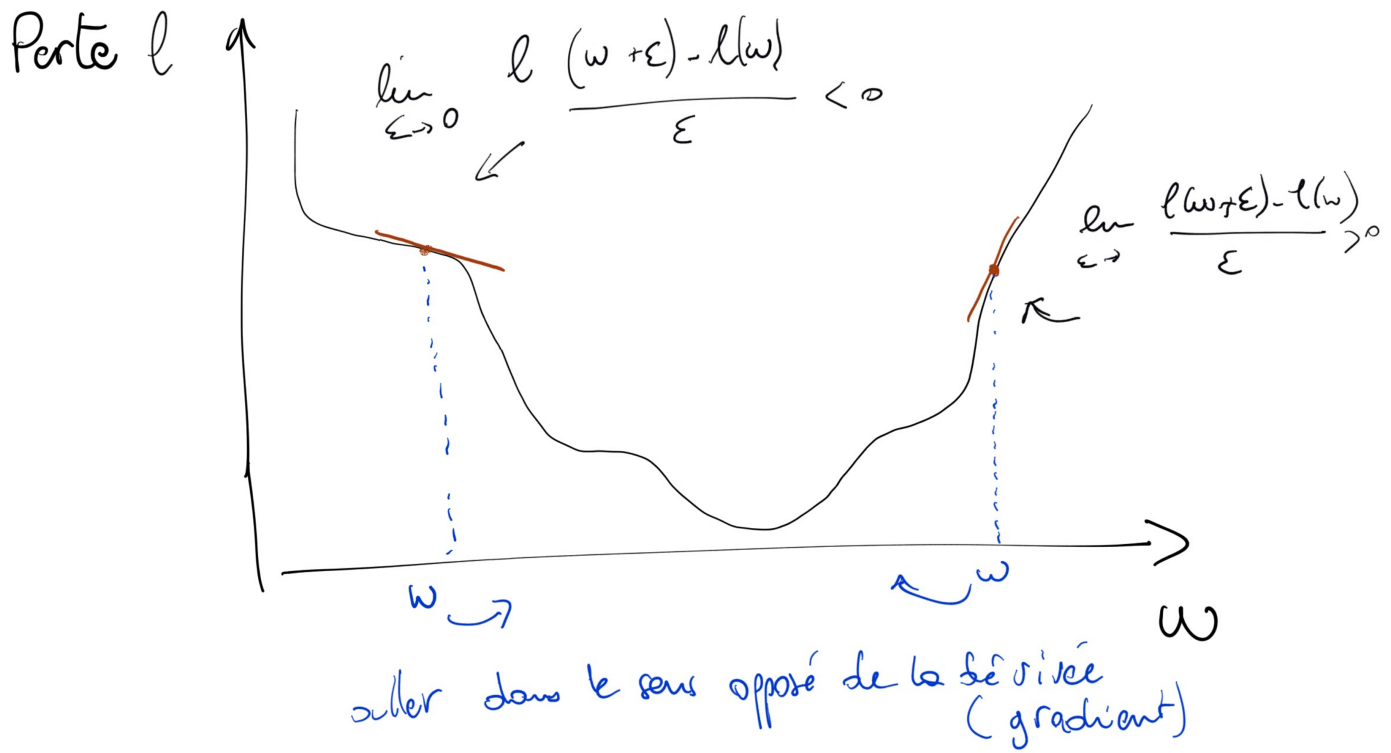
Sens de la dérivée



Rappel de dérivée

Descente de gradient III

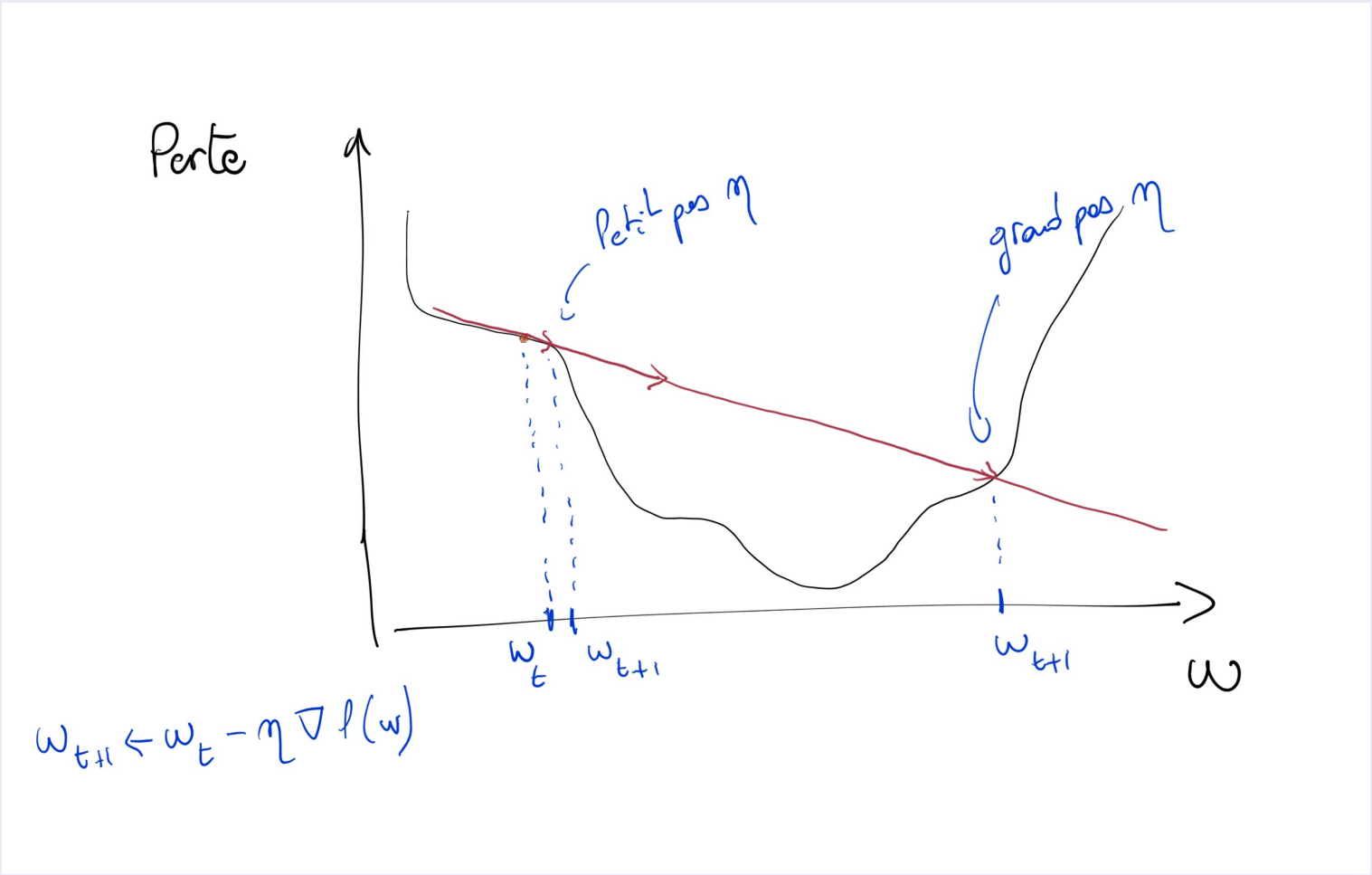
Direction



Le gradient donne la direction : à l'opposé du gradient

Descente de gradient IV

Pas à pas



On avance d'un pas dans cette direction, d'un facteur à préciser, le pas d'apprentissage (learning rate).

Algorithme

- On démarre avec des poids aléatoires, ce qui définit un point sur la courbe de la fonction objectif
- On calcule la dérivée (gradient) de la fonction objectif en ce point
- On avance d'un pas dans la direction opposée à ce gradient. La longueur de ce pas est déterminée par le learning rate.
- On utilise un contrôle de terminaison sur la norme du gradient (tolerance) ou sur l'erreur.
- Remarques
  - Dans le cas convexe (comme pour MSE) on a la garantie d'avoir un minimum global
  - Il est préférable d'avoir des dimensions normalisées (utiliser par exemple `StandardScaler`).
  - Pour le pas de gradient on peut faire du simulated annealing (refroidir lentement) la valeur de learning rate descend lentement (de l'ordre de $1/t$)

Batch et stochastique

Gradient descent (Batch Gradient Descent)

- L'algorithme demande de passer à chaque itération sur l'ensemble des données (rappel : la fonction de perte est une somme sur tous les exemples)
- Le coût de ce calcul peut être encore prohibitif
- Parfois difficile de se sortir de minima locaux

Gradient Stochastique (SGD)

- À chaque étape on sélectionne un seul exemple
- Avantages
  - rapidité
  - si la fonction de perte est très instable pour sortir de minima locaux au caractère stochastique
- Remarque
  - Variante mini batch : on sélectionne un sous-ensemble des exemples
  - adaptée aux GPU car on fait maintenant de petites opérations matricielles en prenant un batch de données

Régression polynomiale

- On a construit des modèles qui sont linéaires (droite, hyper-plans...).
- Cette classe d'hypothèses est peut être inadaptée aux données qui ont un caractère non linéaire
- On peut contourner ce problème en ajoutant de nouveaux attributs aux données : le carré, le cube, des attributs
- On va résoudre

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_k x^k$$

- Les données sont transformées mais on applique toujours le même algorithme de régression linéaire !
- La même chose s'applique si x est un vecteur dans $\mathbb{R}^d$ et on peut faire des produits entre différents attributs ($x_i x_j$, $x_i x_j x_k$...).
- NB : on peut éviter de matérialiser la construction de tous ces nouveaux attributs par des techniques de noyaux (utilisées dans les SVMs, ...).

Techniquement

- avec `PolynomialFeatures`.
- on peut utiliser les objets de classe `Pipeline` pour composer différentes étapes
  - une transformation des features

    ```
    estimators = [('reduce_dim', PCA()), ('clf', SVC())]
    pipe = Pipeline(estimators)
    # deux _ pour acceder aux params
    pipe.set_params(clf__C=10)
    ```
  - l'application d'un modèle avec `pipe.fit`, (`score`, `predict`,...)

Décomposition de l'erreur

Rappel : Biais complexité/variance

- biais : hypothèse sur la classe de fonctions par exemple. Sujet à underfitting si le biais est trop fort
- variance/complexité : variation dans la classe de fonction due à la sensitivité par rapport aux données. Sujet à overfitting si la variance est trop forte
- NB : reste toujours aussi une erreur irréductible due à la qualité des données dans la décomposition de l'erreur

Décomposition de l'erreur

- $L_{\mathcal{D}}(h_S)$ est la somme de l'erreur d'approximation + l'erreur d'estimation
- Erreur d'approximation
  - Ne dépend pas de S mais du choix de $\mathcal{H}$.
  - Diminue si on complexifie $\mathcal{H}$
- Erreur d'estimation
  - diminue si on possède plus d'exemples dans S (on réduit la variance...)

Illustration sur la régression polynomiale

- la technique d'ajouter de nombreux attributs augmente la complexité, la variance.
- le degré des p polynômes, x_i, x_i^2, ...x_i^p ; les tailles des produits $x_i x_j$, $x_i x_j x_k$, ... sont des hyper-paramètres à régler pour contrôler biais/complexité.
- il faut tenter de contrôler cela :
  - tracé des courbes d'apprentissage
  - régularisation

Méthodologie

- Il faut régler les hyper-paramètres sur l'échantillon d'apprentissage.

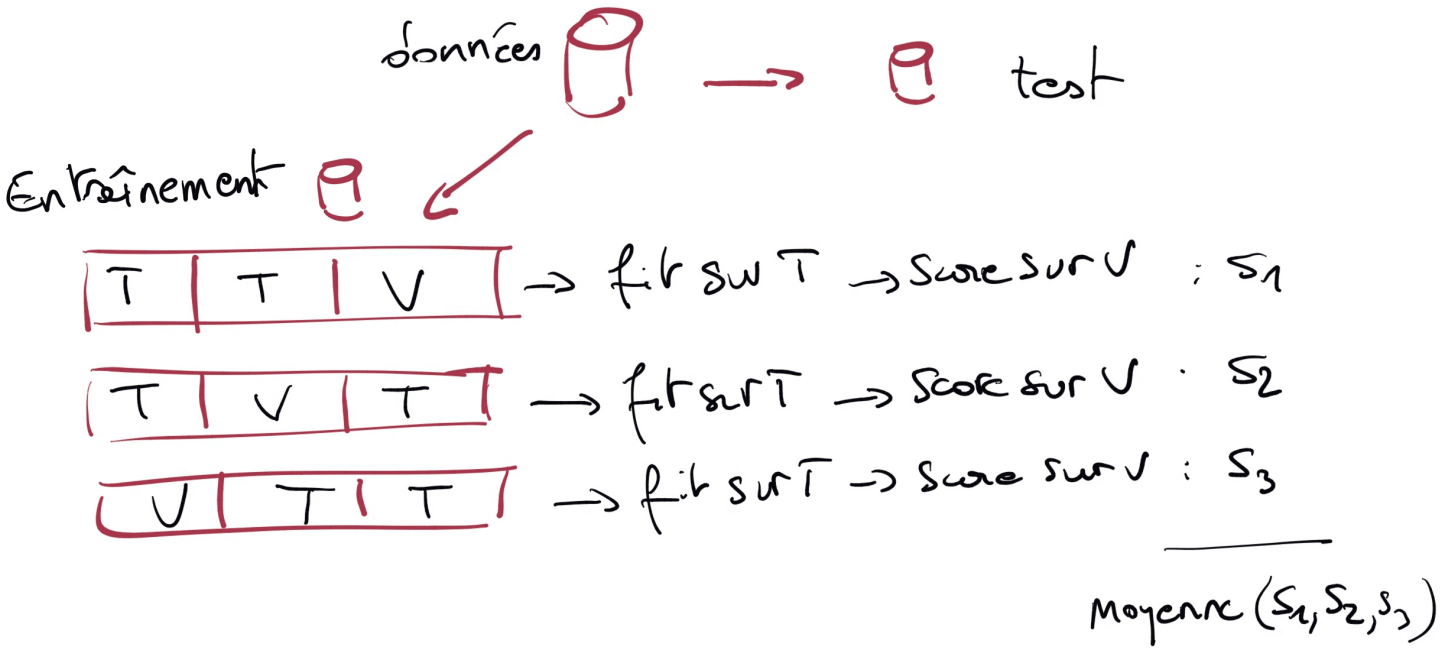
Retenir

- On ne doit jamais regarder l'ensemble test, sauf pour évaluer l'erreur réelle/en généralisation.

Techniques de sélection de modèles

- Découpage de l'ensemble d'entraînement en apprentissage et validation ou
- validation croisée,
- leave-one out,...
- https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

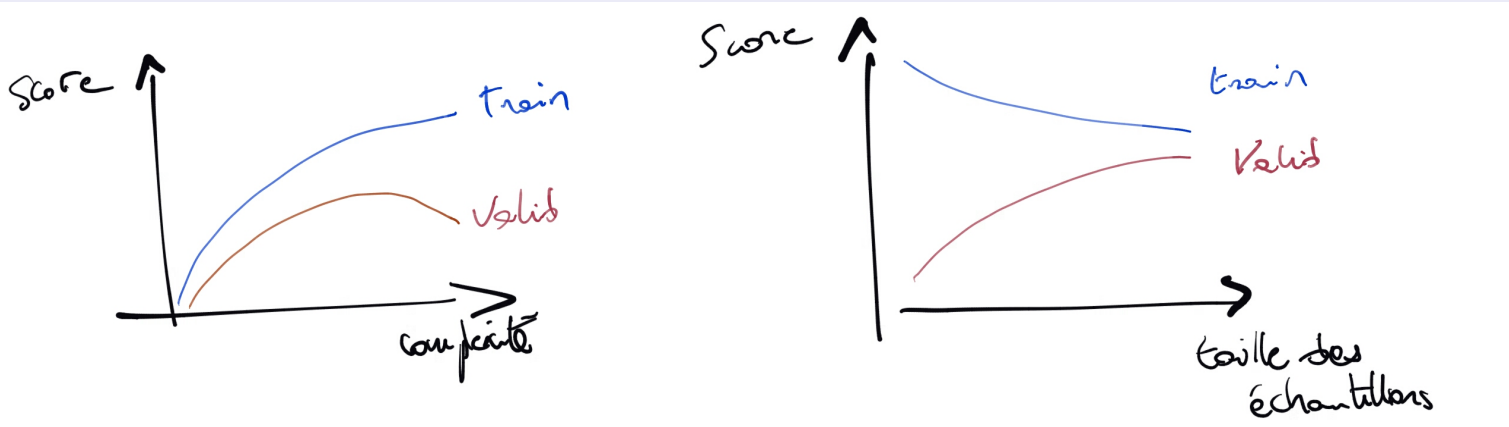
Validation croisée



En pratique

Courbes d'apprentissage

- faire le calcul de l'erreur en train et validation selon la taille de l'échantillon ou la complexité du modèle.
- l'examen de la courbe peut contribuer à sélectionner les bons paramètres



Grid Search

- faire varier l'ensemble des hyper-paramètres et faire une recherche exhaustive du meilleur jeu de paramètres

Régularisation I

Ridge Regression

- Tikonov regularization : ajoute le carré de la norme du vecteur des paramètres ($+\alpha \frac{1}{2} \sum_i w_i^2$). C'est aussi appelé une pénalisation ℓ_2
- Il vaut mieux normaliser les données avant d'appliquer cette régularisation
- On augmente le bais et réduit la variance
- Une forme close est calculable, mais on peut aussi appliquer du SGD.
- Dans sklearn la classe `Ridge` ou `SGDRegressor` avec `penalty` qui vaut `"l2"` avec le paramètre `alpha`.

Régularisation II

Lasso

- Cette fois c'est la norme ℓ_1, la somme des valeurs absolues ($\alpha \sum_i |w_i|$).
- Permet de supprimer les attributs les moins pertinents : obtention d'un modèle parcimonieux par sélection de variable.
- La fonction n'est pas dérivable en 0 (dérivée à droite différente de la dérivée à gauche).
- On peut utiliser une sous-différentielle (sous-gradient, subgradient) autour de 0 en prenant le signe si w est non nul et 0 sinon.
- Classe `Lasso` avec le paramètre `alpha`

Elastic net

- c'est la somme convexe des deux régularisations (un ratio s'applique entre les 2).
- Classe `ElasticNet` avec les paramètres `ratio` et `alpha`.

Régularisation III

Early stopping

- On limite le nombre d'itération. C'est une forme de régularisation « algorithmique ».
- On s'arrête quand l'erreur en validation remonte.
- Pas facile à identifier car les courbes ne sont pas lisses !
- paramètre `earlystopping` dans les algos de type SGD.
- peut être simulé aussi avec le paramètre `warm_start` qui permet de poursuivre une optimisation au point où on s'était arrêté (avec aussi `max_iter` pour fixer le nombre d'itérations).
- `SGDRegressor` avec une régularisation nulle on prend `penalty=None`.

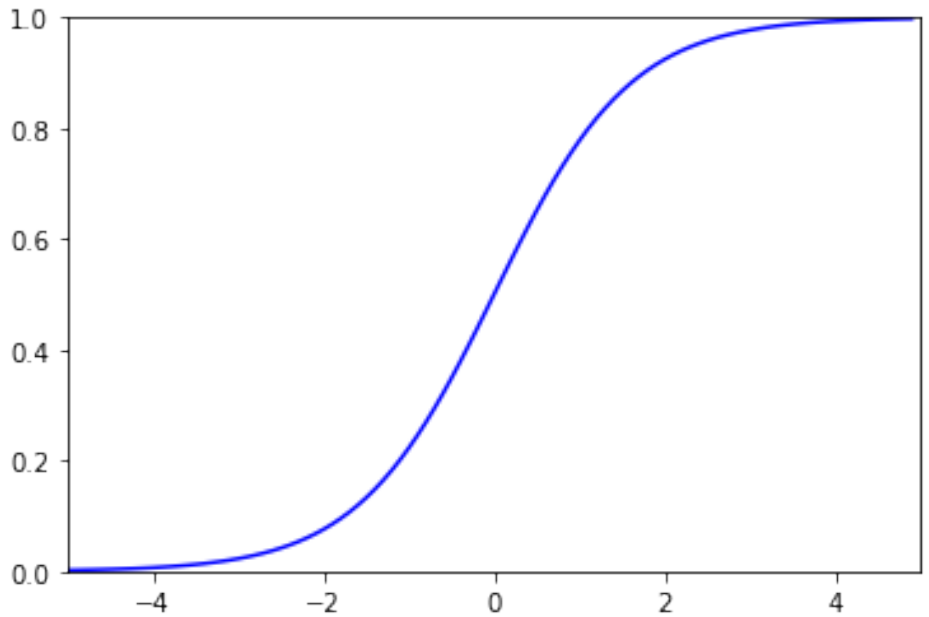
Régression et classification

- Problème de classification :
  - prédire une valeur dans un ensemble fini $\mathcal{Y}$
  - en général la cardinalité de $\mathcal{Y}$ est petite
  - les valeurs de $\mathcal{Y}$ ne sont pas supposées ordonnées
- Beaucoup de méthodes font de la classification binaire ($\mathcal{Y} = \{-1, +1\}$).
- Si la cardinalité de $\mathcal{Y}$ est plus grande que 2 : problème multi-classe
- De nombreuses méthodes peuvent avec quelques variation servir en régression ou en classification
  - Exemple : prédire grâce à $\text{sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$

Régression logistique

- La régression logistique n'est pas une régression mais une classification.
- On cherche à estimer une probabilité d'avoir telle ou telle classe.
- On utilise la fonction logistique sigmoïde :

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}) \text{ avec } \sigma(x) = \frac{1}{1+\exp(-x)}$$



- C'est une fonction continue dérivable entre 0 et 1 et passe par 0.5 en 0.
- Le seuil de décision est 1/2.

Identité

$$
\begin{aligned}
h_{\boldsymbol{w}}(\boldsymbol{x}) &= \frac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})} \\
1 - h_{\boldsymbol{w}}(\boldsymbol{x}) &= 1 - \frac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})} \\
&= \frac{\exp(-\boldsymbol{w}^\top \boldsymbol{x})}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})} \\
&= \frac{1}{1+\frac{1}{\exp(-\boldsymbol{w}^\top \boldsymbol{x})}} \\
&= \frac{1}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x})}
\end{aligned}
$$

Remarque : $\log(h_{\boldsymbol{w}}(\boldsymbol{x})) = -\log(1+\exp(-\boldsymbol{w}^\top \boldsymbol{x}))$

Entraînement : explication par la vraisemblance

- On a l'estimation de la probabilité d'avoir $y = +1$ qui est $h_{\boldsymbol{w}}(\boldsymbol{x})$ et $y = -1$ qui est $1 - h_{\boldsymbol{w}}(\boldsymbol{x})$.
- L'idée est trouver le $\boldsymbol{w}$ qui va maximiser la vraisemblance d'observer notre échantillon.
- Donc on maximise pour chaque couple $(\boldsymbol{x}, y)$, l'expression ($\mathbb{1}_t$ vaut 1 si t est vrai et 0 sinon).

$$h_{\boldsymbol{w}}(\boldsymbol{x})^{\mathbb{1}_{y=+1}}(1 - h_{\boldsymbol{w}}(\boldsymbol{x}))^{\mathbb{1}_{y=-1}}$$

- On prend souvent le logarithme de la vraisemblance (log-vraisemblance), pour faciliter le calcul

$$\mathbb{1}_{y=+1} \log(h_{\boldsymbol{w}}(\boldsymbol{x})) + \mathbb{1}_{y=-1} \log(1 - h_{\boldsymbol{w}}(\boldsymbol{x}))$$

- Ce qui revient à minimiser l'opposé, qu'on désigne par la perte :

$$\begin{aligned}
\ell(h_{\boldsymbol{w}}, (\boldsymbol{x}, y)) &= -\mathbb{1}_{y=+1} \log(h_{\boldsymbol{w}}(\boldsymbol{x})) - \mathbb{1}_{y=-1} \log(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \\
&= \mathbb{1}_{y=+1} \log(1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})) + \mathbb{1}_{y=-1} \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x})) \\
&= \log(1 + \exp(-y\boldsymbol{w}^\top \boldsymbol{x}))
\end{aligned}$$

Entraînement : raccourci intuitif

- On veut que $h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^\top \boldsymbol{x})}$ soit grand quand $y = 1$ (et $1 - h_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+\exp(\boldsymbol{w}^\top \boldsymbol{x})}$ grand si $y = -1$.
- On veut donc qu'une perte grandisse avec $\frac{1}{1+\exp(y\boldsymbol{w}^\top \boldsymbol{x})}$ ou $1 + \exp(-y\boldsymbol{w}^\top \boldsymbol{x})$
- la fonction de perte associée est le log de cette quantité

$$\ell(h_{\boldsymbol{w}}, (\boldsymbol{x}, y)) = \log(1 + \exp(-y\boldsymbol{w}^\top \boldsymbol{x}))$$

- On applique le principe ERM pour trouver $\boldsymbol{w}$.
- C'est convexe ! mais il n'y a pas de forme close mais on utilise la descente de gradient pour trouver le minimum.
- on peut appliquer les mêmes types de régularisation qu'en régression linéaire
- C'est la classe `LogisticRegression` (le paramètre de régularisation est C, une grande valeur de C provoque une faible régularisation).

Problèmes multi-classes

- un jeu de paramètres par classe : $s_k(\boldsymbol{x}) = (\boldsymbol{w}^{(k)})^\top \boldsymbol{x}$ donne un score par classe.
- on veut des probabilités donc compris entre 0 et 1 et la somme sur toutes les classes vaut 1 :

$$p_k = \frac{\exp(s_k(\boldsymbol{x}))}{\sum_j \exp(s_j(\boldsymbol{x}))}$$

- la plus grande valeur est fortement renforcée grâce à l'exponentielle : version continue du calcul d'un max, le softmax
- on prédira ensuite le k qui maximise ce softmax.
- dans le cas binaire, on retrouve $h_{\boldsymbol{w}}$ en posant $\boldsymbol{w} = \boldsymbol{w}^{(1)} - \boldsymbol{w}^{(2)}$ (exercice).
- la classe `LogisticRegression` avec le paramètre `multi_class`.