TP : Régression & Machine Learning

Résumé

L'objectif de ce TP est de poursuivre l'étude du problème de regression linéaire initiée en TD de statistiques, tout en abordant les connections avec le domaine du *machine learning*. Nous verrons en particulier comment utiliser les principes de la régression linéaire pour réaliser des tâches de classification, qui est un des problèmes emblématiques du machine learning.

1 Introduction

Dans le contexte de l'apprentissage supervisé en *machine learning*, on dispose d'une base d'apprentissage

$$((\mathbf{x}_i, y_i))_{i=1,\dots,n},$$

où $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^p$ sont les données (ou features) et où $y_1, \dots, y_n \in \mathcal{Y}$ sont les étiquettes (ou labels) associés. On cherche alors à "apprendre" une fonction f telle que

$$y_i \approx f(\mathbf{x}_i),$$

afin de pouvoir ensuite prédire l'étiquette inconnue d'une nouvelle donnée $\mathbf{x}$. Dans le cas où l'espace des étiquettes est $\mathcal{Y} = \mathbb{R}$, on parle de *régression* tandis que si $\mathcal{Y} = \{-1, 1\}$ (respectivement $\mathcal{Y} = \{1, \dots, k\}$, $k > 2$), on parle de *classification binaire* (respectivement *classification multi-classes*). Notons que les tâches de classification peuvent être vues comme un cas particulier de la régression.

La fonction f est classiquement estimée en minimisant un *risque empirique* parmi une classe de fonctions $\mathcal{F}$ appelée *espace des hypothèses* :

$$\hat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} R(f), \tag{1}$$

avec

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(f(\mathbf{x}_i), y_i\right) + r(f),$$

et où ℓ est une *fonction de perte* et $r(f)$ un terme de *régularisation* ne dépendant que de la fonction et indépendant des données en jeu. Le data scientist doit alors choisir au mieux l'espace $\mathcal{F}$ et les fonctions ℓ, r en prenant en compte la nature des données en jeu.

2 Régression OLS et Ridge

Dans cette partie, on considère l'espace $\mathcal{F}$ des fonctions $f : \mathbb{R}^p \to \mathbb{R}$ du type

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{x},$$

avec $\beta_0 \in \mathbb{R}$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T \in \mathbb{R}^p$, qui peut être canoniquement assimilé à l'espace $\mathbb{R}^{p+1}$. Par la suite, on notera donc $R(\beta_0, \boldsymbol{\beta})$ et $r(\beta_0, \boldsymbol{\beta})$ en lieu et place de $R(f)$ et $r(f)$.

OLS. L'approche par moindres carrés (ou OLS pour Ordinary Least Squares), consiste à choisir comme fonction de perte $\ell(u, v) = (u - v)^2$ et comme régularisation $r(\beta_0, \boldsymbol{\beta}) = 0$. C'est cette approche que nous avons abordée en TD et on rappelle que les coefficients optimaux $\hat{\beta}_0, \hat{\boldsymbol{\beta}}$ (i.e. solutions de (1)) sont donnés par

$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\boldsymbol{\beta}} \end{pmatrix} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \tag{2}$$

avec $\mathbf{y} = (y_1, \ldots, y_n)^T$ et

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \ldots & x_{1,p} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_{n,1} & \ldots & x_{n,p} \end{pmatrix}.$$

La classe `sklearn.linear_model.LinearRegression` de `scikit-learn` contient une implémentation de la régression OLS, mais qui peut-être néanmoins recodée sans aucune difficulté.

1. Charger le fichier `data1.npy` et afficher le nuage de points. Mettre en œuvre une régression linéaire OLS et superposer la droite obtenue sur le nuage. Donner l'erreur d'apprentissage résultante (i.e. la valeur de $R(\beta_0, \boldsymbol{\beta})$).
2. Charger le fichier `data2.npy` et refaire les étapes de la question 1. Que peut-on constater ?

Espace de redescription. Quand le modèle étiquettes-données à apprendre est par nature non-linéaire, l'utilisation d'un modèle linéaire entraîne en général des erreurs importantes. Une technique couramment utilisée consiste à appliquer une fonction $\phi : \mathbb{R}^p \mapsto \mathbb{R}^q$ aux données pour les « transporter » dans un *espace de plus grande dimension* $q > p$ où la description par un modèle linéaire sera pertinente. On considère alors plus généralement une classe $\mathcal{F}$ de fonctions du type

$$f(\mathbf{x}) = \sum_{i=1}^{q} \beta_i \phi_i(\mathbf{x}), \tag{3}$$

où $\phi_1, \ldots, \phi_q$ représente les composantes de ϕ. Dans le cas $p = 1$, on retrouve par exemple le modèle linéaire en posant $\phi_i(x) = x$ et on obtient également un modèle polynomial en posant $\phi_i(x) = x^i$ ou encore un modèle sigmoïde en posant $\phi_i(x) = \left(1 + \exp\left(-\frac{x - \mu_i}{\sigma_i}\right)\right)^{-1}$.

3. Etendre l'approche OLS et donner l'expression des coefficients $\hat{\beta}_0, \hat{\boldsymbol{\beta}}$ optimaux pour l'espace $\mathcal{F}$ décrit en (3). Préciser les éventuelles hypothèses nécessaires.
4. Considerer un modèle polynomial et adapter la régression OLS sur le jeu de données `data2.npy` (il faudra ajuster l'ordre q du modèle). Donner l'erreur d'apprentissage et commenter au regard de (2).
5. Charger le jeu de données `data3.npy` et afficher le nuage de points. Ajuster par régression OLS un modèle polynomial d'ordre $q = 10$. Commenter.

Ridge. Il existe plusieurs situations où la matrice $\mathbf{X}$ peut avoir un rang déficient (par exemple $p > n-1$) où plus généralement posséder des lignes proches d'être linéairement dépendantes. Dans ce cas, la solution du problème des moindres carrés n'est plus unique et n'est plus donnée par (2). L'approche ridge consiste alors à introduire une régularisation quadratique en posant $r(\beta_0, \boldsymbol{\beta}) = \lambda(\beta_0^2 + \|\boldsymbol{\beta}\|_2^2)$ où $\lambda > 0$ contrôle le poids donné à la régularisation.

6. Résoudre le problème d'optimisation

$$\min_{(\beta_0,\boldsymbol{\beta})\in\mathbb{R}^{p+1}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \boldsymbol{\beta}^T\mathbf{x}_i)^2 + \lambda\sum_{i=0}^{p}\beta_i^2,$$

   et donner l'expression des coefficients optimaux $\hat{\beta}_0, \hat{\boldsymbol{\beta}}$ de l'approche ridge. On prendra soin de bien justifier chaque étape du calcul.

La classe `sklearn.linear_model.RidgeClassifier` de `scikit-learn` contient une implémentation de la régression ridge, mais celle-ci peut être facilement recodée comme pour OLS.

7. Reprendre le jeu de données `data3.npy` et comparer les approches OLS et ridge. Commenter suivant le choix du paramètre de régularisation λ.

LASSO. Le LASSO (pour Least Absolute Shrinkage and Selection Operator) est une approche proposée par Tibshirani en 1996, adaptée au cas d'un modèle parcimonieux (peu de coefficients non-nuls dans la solution). On utilise une régularisation ℓ^1 associée à la fonction de perte quadratique et on cherche à résoudre le problème d'optimisation

$$\min_{(\beta_0,\boldsymbol{\beta})\in\mathbb{R}^{p+1}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \boldsymbol{\beta}^T\mathbf{x}_i)^2 + \lambda\sum_{i=0}^{p}|\beta_i|,$$

où le coefficient $\lambda > 0$ contrôle le poids donné à la régularisation. Il n'y a pas en général de solution analytique à ce problème et la fonction de coût (qui est convexe et non-différentiable) doit être minimisée numériquement à l'aide d'algorithmes dédiés. Pour une implémentation du LASSO sous `scikit-learn`, voir `sklearn.linear_model.Lasso`.

8. Reprendre le jeu de données `data3.npy` et ajouter le LASSO en comparaison des approches OLS et ridge. Commenter l'impact de la régularisation ℓ^1.

3 Classification

Classification. La classification est un des problèmes standards étudiés en machine learning. Étant donné un vecteur de données $\mathbf{x} \in \mathbb{R}^p$ pouvant appartenir à k classes, on cherche à construire une fonction de classification (ou *classifieur*)

$$f : \mathbb{R}^p \mapsto \{1, \dots, k\}.$$

Le classifieur f sera construit à partir à partir d'exemples « étiquettés » $(\mathbf{x}_i, y_i)_{i=1,\dots,n}$ où $y_i \in \{1, \dots, k\}$ représente l'indice de la classe associée à la donnée $\mathbf{x}_i$.

Une famille de classifieurs couramment utilisée est celle des *classifieurs linéaires*. Dans le cas binaire, et en supposant les étiquettes à valeurs dans $\{-1, 1\}$, ils sont donnés par la classe $\mathcal{F}$ de fonctions de la forme

$$f(\mathbf{x}) = \operatorname{sgn}\left(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}\right) = \begin{cases} 1 & \text{si } \beta_0 + \boldsymbol{\beta}^T\mathbf{x} \geq 0 \\ -1 & \text{si } \beta_0 + \boldsymbol{\beta}^T\mathbf{x} < 0 \end{cases},$$

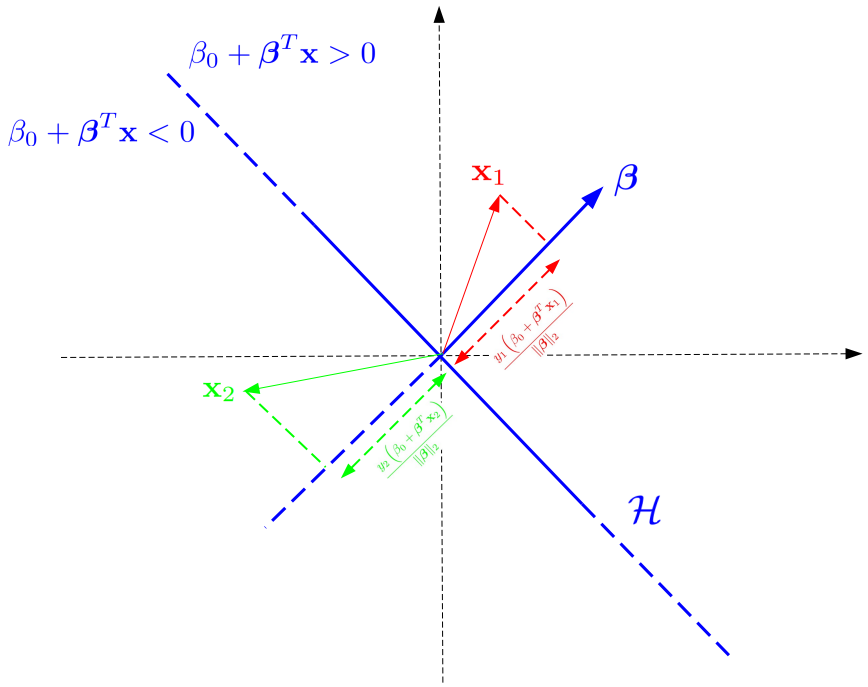

FIGURE 1 – Géométrie d'un classifieur linéaire

avec $\beta_0 \in \mathbb{R}$ un terme de biais (ou *intercept*), et $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$ le vecteur des coefficients.

La géométrie d'un classifieur linéaire est illustrée en Figure 1. En particulier, on observe que la frontière de décision $\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{\beta}^T\mathbf{x} = 0\}$ est un hyperplan affine et on montre facilement que la distance entre une donnée $\mathbf{x}$ et l'hyperplan s'écrit

$$d(\mathbf{x}, \mathcal{H}) = \frac{m(\mathbf{x}, f(\mathbf{x}))}{\|\boldsymbol{\beta}\|_2},$$

où $m(\mathbf{x}, y) = y\left(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}\right)$ est la *marge* associée à $\mathbf{x}$ et son étiquette y. Afin d'apprendre les paramètres $\beta_0, \boldsymbol{\beta}$ à partir d'exemples $((y_i, \mathbf{x}_i))_{i=1,\ldots,n}$, on minimise en général un risque du type

$$R(\beta_0, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(m(\mathbf{x}_i, y_i)\right) + r\left(\beta_0, \boldsymbol{\beta}\right), \tag{4}$$

avec $\ell : \mathbb{R} \mapsto \mathbb{R}^+$ une fonction de perte et $r : \mathbb{R}^{p+1} \mapsto \mathbb{R}^+$ une fonction de régularisation. En particulier, le risque R ne dépend des exemples qu'à travers leurs *marges* $y_i\left(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i\right)$.

Afin d'étendre la classification du cas binaire au cas multiclasses ($k > 2$), plusieurs options sont possibles :

— *One-vs-Rest.* On se ramène ici au cas binaire en entraînant k classifieurs binaires $f_1, \ldots, f_k$ (un pour chaque classe). Par exemple, pour la classe i, le classifieur f_i est entraîné en considérant les exemples de la i-ème classe comme positifs et ceux des $k-1$ classes restantes comme négatifs. Le classifieur final est donné par

$$f(\mathbf{x}) = \underset{i=1,\ldots,k}{\operatorname{argmax}}\, f_i(\mathbf{x}).$$

— *One-vs-One.* On entraine ici $\binom{k}{2}$ classifieurs binaires en considérant les exemples de deux classes pour l'entraînement. Le résultat de la classification est ensuite donné par un vote à majorité.

On se référera à la documentation de `scikit-learn` pour connaître les options utilisées.

Données. On fournit dans le fichier `classif.py` un code pour la génération de données dans le cas $p = 2$. Il permet en particulier de générer des données de deux classes différentes basées sur la loi gaussienne multivariée. Il inclut également la génération d'*outliers* (données aberrantes) pour tester la robustesse des méthodes de classification. Différentes matrices de covariance permettant de modifier la forme des nuages de points peuvent être utilisées.

Classifieurs OLS/ridge. On adapte ici les approches OLS et ridge en régression pour la tâche de classification binaire.

1. Montrer que dans le cas d'étiquettes à valeurs dans $\{-1, 1\}$, les risques (1) pour la régression OLS et ridge peuvent se réécrire comme (4) où l'on identifiera la nouvelle fonction de perte ℓ.
2. Générer des données à l'aide du fichier `classif.py`. Afficher le nuages de points et ajouter les classifieurs OLS/ridge en traçant leurs hyperplans séparateurs. Tester les résultats pour différentes valeurs de taille d'échantillons n_0, n_1. Commenter.

Régression logistique. La régression logistique consiste à choisir dans (4) la fonction de perte logistique donnée par

$$\ell(m) = \log\left(1 + \mathrm{e}^{-m}\right).$$

On choisira également une régularisation ℓ^2. Le classifieur est implémenté dans `scikit-learn` sous la classe `sklearn.linear_model.LogisticRegression`.

3. Discuter de l'intérêt de la fonction de perte logistique par rapport à la fonction de perte quadratique.
4. Ajouter le classifieur par régression logistique aux expérimentations précédentes et commenter les performances au regard des deux classifieurs précédents.
5. Augmenter le nombre d'outliers n_{out} et commenter les performances des trois classifieurs.

4 Tests sur la base de données MNIST

Dans cette partie, nous considérons le cas d'une classification multi-classes et testons les performances de la régression logistique sur la base `MNIST` composée d'images de chiffres manuscrits, dont un exemple est donné en FIGURE 2. Les données sont fournies avec leurs étiquettes à valeurs dans $\{0, \ldots, 9\}$. Le fichier `mnist.py` fournit des briques de code permettant de charger la base de données et d'afficher quelques exemples.

1. Entraîner le classifieur par régression logistique avec régularisation ℓ^2 sur la base d'apprentissage formées des variables `X_train` et `y_train`. On normalisera au préalable les données en utilisant la classe `sklearn.preprocessing.StandardScaler`. A l'aide de la fonction `sklearn.metrics.confusion_matrix` et de la classe `sklearn.metrics.ConfusionMatrixDisplay`, afficher la matrice de confusion.

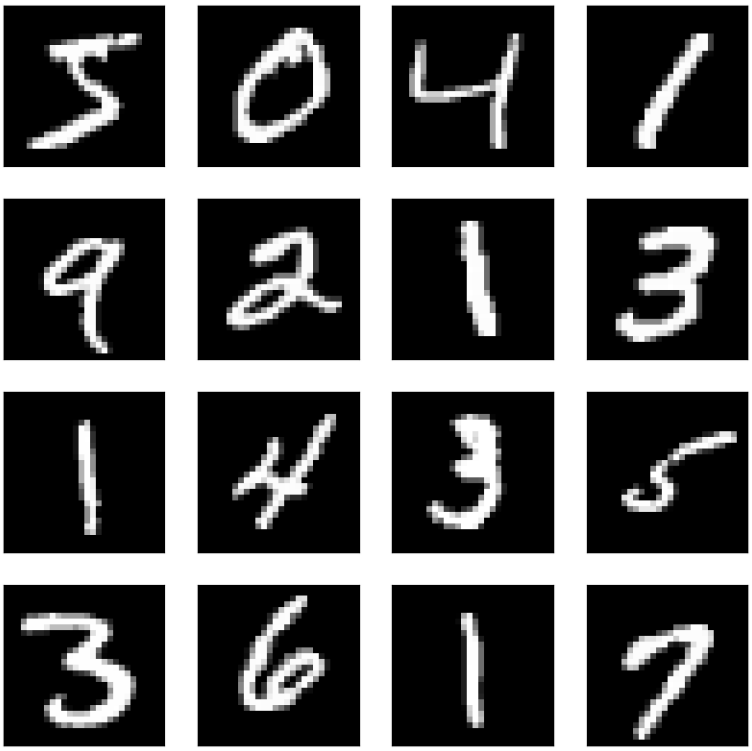

FIGURE 2 – Echantillon de la base MNIST

2. Afficher les coefficients $\hat{\boldsymbol{\beta}}$ sous forme d'image pour chaque classe (utiliser une carte de couleur `RdBu` pour la visualisation. Commenter les résultats.
3. On s'intéresse à présent à la régularisation ℓ^1, donnée par la fonction

$$r(\beta_0, \boldsymbol{\beta}) = \sum_{i=0}^{p} |\beta_i|.$$

Réentraîner le classifieur par régression logistique et comparer les images des coefficients pour chaque classe avec le cas d'une régularisation ℓ^2. Commenter.