Machine learning — TP 1

Cycle pluridisciplinaire d'études supérieures
Université Paris sciences et lettres

Joon Kwon

vendredi 13 mars 2020

1 Premiers pas avec NumPy et Matplotlib

1.1 NumPy

NumPy introduit le type de données `array`, qui est un tableau multidimensionnel d'éléments. Un `array` de dimension 1 correspond à un vecteur et un `array` de dimension 2 à une matrice. Pour se familiariser avec le package, on pourra exécuter chaque ligne de code ci-dessous, observer le résultat et éventuellement effectuer des manipulations supplémentaires.

On charge le package.

```
import numpy as np
```

Une façon de contruire un `array` de dimension 1 est la suivante.

```
a = np.array([1,42,18])
a
```

La première ligne permet de créer l'`array` et la seconde permet d'afficher son continu.

Voici d'autres façons de construire des `array` de dimension 1 pour lesquels on pourra également observer le résultat.

```
a = np.array([1,42,18])
b = np.arange(10)
c = np.arange(2,5,.5)
c = np.linspace(0,1,11)
d = np.ones(6)
d = np.zeros(5)
e = np.full(5,3)
```

Voici deux façons différentes de créer la même `array` de dimension 2.

```
A = np.array([[1,2,3], [4,5,6], [7,8,9]])
B = np.array([1,2,3,4,5,6,7,8,9]).reshape((3,3))
```

`np.ones`, `np.zeros`, `np.full` et `np.random.random` peuvent prendre un `tuple` en argument pour renvoyer des `array` multidimensionnel.

```
C = np.ones((2,5))
```

On a accès à différentes informations sur un `array`.

```
A.ndim
A.size
A.shape
```

On peut créer des `array` en combinant plusieurs `array`.

```
np.concatenate((a,b))
np.vstack((A,a))
np.hstack((A,A))
```

Les opérations sur les `array` se font élément par élément et renvoient un `array`.

```
f = np.array([2,3])
g = np.array([1.5, 3])
f+g
f*g
f**g
3*f
f == g
f < g
f <= g
```

Pour la multiplication matricielle, il faut utiliser une des syntaxes suivantes.

```
A.dot(a)
np.dot(A,a)
A@a
```

On prend la transposée d'une matrice de la façon suivante.

```
A.T
```

Il existe un certain nombre de fonctions qui s'appliquent soit à l'ensemble des éléments d'un `array`, ou bien selon une dimension.

```
A.sum()
A.min()
A.max()
A.mean()
A.sum(axis=0)
A.sum(axis=1)
```

On peut accéder aux éléments d'un `array` à l'aide des indices, qui commencent par défaut à 0.

```
a
a[0]
A
A[1,1]
A[1]
```

On peut aussi spécifier des sous-ensembles d'indices.

```
b
b[:]
b[1:8]
b[1:8:2]
b[1:8:2][2]
b[::-1]
A
A[:,1]
```

On peut également spécifier un ensemble d'indices à l'aide d'une liste.

```
a
a[[0,1,1]]
```

On peut aussi utiliser une liste de booléens (`True`, `False`) pour spécifier si on sélectionne ou non chaque élément (une version récente de NumPy est nécessaire pour cette fonctionnalité).

```
a[[False, True, True]]
```

On peut enfin sélectionner les valeurs qui vérifient une condition.

```
b[b>5]
```

On peut utiliser un `array` comme ensemble de valeurs à parcourir dans une boucle `for`.

```
for i in a:
    print(i**2)
```

NumPy propose un grand nombre de fonctions qui s'appliquent aux `array` composante par composante.

```
np.cos(1)
np.cos(a)
```

1.2 Matplotlib

Matplotlib est une librairie permettant de créer des visualisations de données. Pour les tâches qui vont nous occuper, il suffit de charger `matplotlib.pyplot`.

```
import matplotlib.pyplot as plt
```

Une des tâches les plus courantes est de tracer le graphe d'une fonction. Il faut pour cela se donner un `array` de dimension 1 contenant des abscisses. On définit ci-après un `array` `x` contenant 10 nombres à intervalles réguliers sur $[0, 4]$.

```
x = np.linspace(0,4,10)
```

Pour chaque valeur dans `x`, il faut se donner l'ordonnée correspondante à la fonction que l'on souhaite tracer.

```
y = np.sin(x)
```

On peut à présent générer la figure.

```
plt.plot(x, y)
plt.show()
```

On observe que le graphe est simplement donné par un ensemble fini de points reliés par des segments. Afin d'obtenir un graphe plus lisse, il convient d'augmenter le nombre d'abscisses en lesquels on calcule la fonction. On peut également tracer plusieurs fonctions sur une même figure.

```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

On peut personnaliser le style du tracé de la fonction en ajoutant un troisième argument de type chaîne de caractères à la fonction `plt.plot`. Essayer par exemple `'g--'` ou encore `'-.k'`. On peut souhaiter fixer manuellement les limites des axes ; on procède alors comme suit.

```
plt.plot(x, np.sin(x))
plt.axis([0, 5, -0.5, 0.5])
plt.show()
```

Dans d'autres situations, les abscisses et les ordonnées que l'on fournit ne représentent pas une fonction, mais simplement un ensemble de points. On souhaite alors les représenter sans qu'ils ne soient reliés par des segments. On utilise pour cela la fonction `plt.scatter` au lieu de la fonction `plt.plot`.

```
plt.scatter([2,1,3],[-1,-1,0])
plt.show()
```

Enfin, on peut ajouter différents éléments pour annoter la figure.

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.title('une courbe sinusoïdale')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.legend()
plt.show()
```

Il ne faut pas oublier l'invoquer la fonction `plt.legend` qui affiche la légende des courbes qui ont été fournies via l'option `label` dans la fonction `plt.plot`.

2 Un problème de régression en dimension 1

On se place dans un cadre de régression en dimension 1, c'est-à-dire l'ensemble des entrées est $\mathcal{X} = \mathbb{R}$ et l'ensemble des sorties $\mathcal{Y} = \mathbb{R}$. Les prédicteurs sont donc des fonctions de la forme $f : \mathbb{R} \to \mathbb{R}$.

2.1 Génération des données

Dans ce TP, nous n'allons pas travailler avec des données réelles, mais nous allons les générer nous-mêmes. On considère la fonction polynomiale :

$$g(x) = \frac{3}{2}x^3 - x^2 - \frac{3}{4}x + 1$$

qu'on va utiliser pour générer un échantillon d'apprentissage $(x_i, y_i)_{i\in[n]}$ i.i.d. selon :

$$x_i \sim \text{Unif}([-1, 1]) \tag{1}$$

$$y_i = g(x_i) + \frac{1}{20}\zeta_i \tag{2}$$

où $\zeta_i \sim \mathcal{N}(0, 1)$ est indépendant de x_i.

Question 1. — Définir un `array` `x` contenant $n = 15$ réels tirés uniformément sur $[-1, 1]$. On pourra utiliser la fonction `np.random.random_sample`.

Question 2. — Définir un `array` `y` contenant les valeurs $(y_i)_{i\in[n]}$ définies selon (2). On pourra faire appel à la fonction `np.random.randn`.

Question 3. — Visualiser dans le plan les données obtenues $(x_i, y_i)_{i\in[n]}$ sous la forme de points.

Question 4. — Avec le même procédé, générer un échantillon de test $(x'_i, y'_i)_{i\in[n']}$ de taille $n' = 30$ qu'on stockera dans des `array` `x_test` et `y_test`.

2.2 Régression linéaire

Nous allons faire appel à la bibliothèque `scikit-learn` pour construire des prédicteurs à partir des données d'apprentissage. Dans `scikit-learn`, tout prédicteur possède une méthode `fit` qui permet de spécifier les données d'apprentissage, ainsi qu'une méthode `predict` qui permet, une fois le prédicteur entraîné, de calculer la prédiction de le prédicteur sur de nouvelles entrées.

On cherche ici à construire le prédicteur linéaire (c'est-à-dire affine) qui minimise l'erreur des moindres carrés. Autrement dit, la classe de prédicteurs considérée est :

$$\mathcal{F} = \{f_{a,b} : x \mapsto ax + b\}_{a,b\in\mathbb{R}},$$

et on cherche à calculer le minimiseur du risque empirique pour la fonction de perte $\ell(y, y') = (y - y')^2$:

$$\widehat{f} = \operatorname*{arg\,min}_{f\in\mathcal{F}} \left\{ \sum_{i=1}^{n} (y_i - f(x_i))^2 \right\}.$$

Nous allons pour cela utiliser le module `LinearRegression` que l'on charge comme suit.

```
from sklearn.linear_model import LinearRegression
f = LinearRegression()
```

On a initialisé ci-dessus un prédicteur `f` de régression linéaire aux moindres carrés. Il faut à présent fournir les données d'apprentissage via la méthode `f.fit`. Celle-ci exige que les données d'entrée $(x_i)_{i\in[n]}$ soient présentés sous la forme d'un `array` de dimension 2 contenant n lignes (une par donnée) et 1 colonne (car ici l'espace d'entrée $\mathcal{X} = \mathbb{R}$ est de dimension 1). Or, `x` comporte une seule dimension. Nous allons donc définir `X` de sorte qu'il contienne les valeurs de `x` sous la bonne forme, et de même pour `x_test` :

```
X = x[:,np.newaxis]
x_test = x_test[:,np.newaxis]
```

On peut à présent spécifier les données apprentissage.

```
f.fit(X,y)
```

Si on écrit $\widehat{f}(x) = \hat{a}x + \hat{b}$, les coefficients $\hat{a}$ et $\hat{b}$ du prédicteur $\widehat{f}$ sont donnés par `f.coef_` et `f.intercept_` respectivement. Les prédictions du prédicteur $\widehat{f}$ sur les données d'apprentissage d'une part, et données de test d'autre part sont donnés par :

```
f.predict(X)
f.predict(x_test)
```

Question 5. — Produire une figure qui superpose aux points $(x_i, y_i)_{i\in[n]}$ la droite correspondant au graphe du prédicteur $\widehat{f}$.

Question 6. — Calculer l'erreur moyenne d'apprentissage ainsi que l'erreur moyenne de test. Commenter.

2.3 Régression polynomiale

Soit $d \geqslant 1$ un entier. On considère à présent la régression aux moindres carrés sur les fonctions polynomiales de degré au plus d. Autrement dit, on considère la classe de prédicteurs :

$$\mathscr{F}_d^{\text{poly}} = \left\{ f_{a_0,\dots,a_d} : x \mapsto \sum_{k=0}^{d} a_k x^k \right\}_{(a_0,\dots,a_d)\in\mathbb{R}^{d+1}}$$

et on souhaite calculer le prédicteur :

$$\widehat{f} := \operatorname*{arg\,min}_{f\in\mathscr{F}_d^{\text{poly}}} \left\{ \sum_{i=1}^{n} (y_i - f(x_i))^2 \right\}.$$

Nous allons voir que ce problème peut être ramené à une régression linéaire en dimension d. On considère l'application $\psi : \mathbb{R} \to \mathbb{R}^d$ définie par :

$$\psi(x) = (x, x^2, \dots, x^d).$$

Pour des coefficients $(a_0, \dots, a_d) \in \mathbb{R}^{d+1}$, notons $\mathbf{a} = (a_1, \dots, a_d) \in \mathbb{R}^d$, et le polynôme correspondant $f_{a_0,\dots,a_d}$ se réécrit alors :

$$f_{a_0,\dots,a_d}(x) = \langle \psi(x), \mathbf{a} \rangle + a_0.$$

Notons $\mathscr{X}' = \mathbb{R}^d$ et $\mathscr{F}'$ la classe des prédicteurs linéaires (c'est-à-dire affines) de $\mathscr{X}'$ dans $\mathbb{R}$:

$$\mathscr{F}' = \left\{ f'_{(a_0,\dots,a_d)} : (z_1, \dots, z_d) \mapsto \sum_{k=1}^{d} a_k z_k + a_0 \right\}.$$

Soit $\widehat{f}'$ le prédicteur obtenu par la régression linéaire des données $(\psi(x_i), y_i)_{i\in[n]} \in (\mathscr{X}' \times \mathbb{R})^n$:

$$\widehat{f}' = \operatorname*{arg\,min}_{f'\in\mathscr{F}'} \left\{ \sum_{i=1}^{n} (y_i - f'(\psi(x_i)))^2 \right\}.$$

On a alors la relation

$$\widehat{f} = \widehat{f}' \circ \psi.$$

Autrement dit, la régression polynomiale (sur $\mathscr{X} = \mathbb{R}$) peut être calculée à l'aide d'une régression linéaire en dimension supérieure (sur $\mathscr{X}' = \mathbb{R}^d$).

Question 7. — On souhaite dans cette question calculer le prédicteur $\widehat{f}$ pour $d = 2$. Nous allons implémenter la fonction ψ introduite ci-dessus à l'aide d'outils fournis par `scikit-learn`.

```
from sklearn.preprocessing import PolynomialFeatures
psi = PolynomialFeatures(2,include_bias=False).fit_transform
```

a) Calculer le prédicteur $\widehat{f}$ en faisant intervenir `LinearRegression()`.
b) Superposer le graphe de $\widehat{f}$ aux points représentant les données d'apprentissage.
c) Calculer les erreurs moyennes d'apprentissage et de test, et les comparer aux erreurs obtenues dans la section précédente.

Question 8. — On souhaite à présent généraliser le calcul à tout $d \geqslant 1$ (qui correspond au degré maximal des polynômes).
a) En s'inspirant de la question précédente, écrire une fonction qui prend en argument l'entier d, et qui renvoie deux éléments : le prédicteur $\widehat{f}$ et la fonction ψ.
b) Calculer le prédicteur pour $d \in \{3, 4, 13, 14\}$ et le visualiser (on pourra si besoin spécifier des limites pour l'abscisse).
c) Pour $d = 3$, comparer les coefficients du prédicteur avec ceux de la fonction g qui a généré les données.
d) Que se passe-t-il pour $d = 14$?
e) Tracer une figure avec en abscisse $d = 1, \dots, 14$, et en ordonnée les graphes des erreurs moyennes d'apprentissage et de test. Commenter.

2.4 Régularisation LASSO

On considère à présent la régression polynomiale aux moindres carrés *avec régularisation LASSO* :

$$\widehat{f} = \operatorname*{arg\,min}_{f \in \mathscr{F}_d^{\text{poly}}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \alpha \sum_{k=1}^{d} |a_k| \right\},$$

où $\alpha > 0$ est un paramètre à choisir.

Avec `sklearn`, on initialise une régression *linéaire* avec régularisation LASSO de la façon suivante.

```
from sklearn.linear_model import Lasso
f = Lasso(0.1)
```

où l'argument correspond au choix du paramètre α.

Question 9. — En s'inspirant de la section précédente, calculer pour $d = 14$, la régression *polynomiale* avec régularisation LASSO pour différentes valeurs du paramètre (par exemple $\alpha \in \{10^{-6}, 10^{-5}, \dots, 10^{-1}, 1\}$), et les visualiser. Que remarque-t-on sur les polynômes obtenus ? sur leurs coefficients ? leurs degrés ?

Question 10. — Tracer les erreurs d'apprentissage et de test pour les différentes valeurs de α. Afin d'améliorer la lisibilité, on pourra utiliser une échelle logarithmique en abscisse et en ordonnée en ajoutant les lignes `plt.xscale('log')` et `plt.yscale('log')` avant `plt.show()`. Commenter.