Travaux pratiques n°1

Programmation Orienté Objet pour le Calcul Scientifique

Sylvain Desroziers

16 octobre 2017

Prérequis

Pour mener à bien ces travaux, il est nécessaire d'avoir sur sa machine :

* un compilateur `C++`, par exemple `g++`,
* les systèmes de compilation `CMake` et `Make`,
* l'outillage d'affichage `gnuplot`.

Remarque : ces outils sont évidemment disponibles sous `MacOS` et `Linux`, il peut toutefois être nécessaire de les installer. A vous de jouer.

1 Les bases de la compilation

Dans cette partie, on s'intéresse à la compilation, c'est-à-dire le processus de transformation du code source écrit par le développeur en du code executable sur la machine.

1. En préambule, écrire un premier programme écrivant la chaîne de caractère `Hello world!` à l'écran.
2. Compiler le programme et lancer l'exécution comme présenté dans le cours.

Compiler un executable à partir d'un seul fichier est trivial. Toutefois, compiler à l'échelle d'un projet réaliste est un processus complexe. En effet, il faut souvent gérer un grand nombre de fichiers. En pratique, les fichiers sont d'abord compilés en fichiers *objets* contenant les instructions en code machine. Ces fichiers intermédiaires sont ensuite assemblés ou *liés*Â pour former l'executable. Pour générer un fichier objet `<file>.o` à partir d'un fichier source `<file>.cpp`, la commande est :

```
desroziers> g++ -o <file>.o -c <file>.cpp
```

et pour générer un executable `exe` :

```
desroziers> g++ <file>.o -o <exe>
```

3. Générer un fichier objet à partir de votre fichier écrit dans 1.
4. Générer ensuite l'executable.

Dans ce cours, on va s'appuyer sur un système de compilation éprouvé pour générer nos solutions logicielles. En particulier, c'est un excellent outil pour se former à la compilation. Ce système est `CMake` (voir `https://cmake.org/` pour la documentation). `CMake` permet d'écrire dans un langage adapté le processus de compilation d'exécutables et de librairies :

```
cmake_minimum_required(VERSION <minimum>) # version de cmake nécessaire

project(<nom du projet> VERSION 1.0 LANGUAGES CXX) # définition d'un projet
# obligatoire

add_executable(<exectuable> # créer un executable
<liste de fichiers sources> # à partir d'une liste de fichiers sources
)
```

`CMake` est un générateur de code à partir de fichier script, cet outil *ne compile pas* les fichiers sources. Par défaut, `CMake` génère des fichiers `Make` (voir `https://www.gnu.org/software/make/` pour la documentation), outil classique que l'on utilisera pour compiler les fichiers sources ensuite. Les commandes `CMake` doivent être écrites dans un fichier script `CMakeLists.txt` placé à la racine du projet et qui va être utilisé par `CMake`. Comme `CMake` va générer du code, une bonne pratique est de ne pas générer dans les sources. On construit alors un répertoire dédié à la compilation, idéalement dans le répertoire racine contenant le fichier `CMakeLists.txt`.

En résumé, les commandes à effectuer sont les suivantes :

```
desroziers> mkdir build
desroziers> cd build
desroziers> cmake ..
desroziers> make
```

5. Ecrire un fichier `CMakeLists.txt` pour générer l'exécutable de la question 1.
6. Procéder à la compilation en analysant les sorties de `CMake`.

`CMake` permet de gérer automatiquement et en ligne de commande les modes de compilation, les installations et plus généralement, le passage d'arguments à vos scripts. :

```
desroziers> cmake .. -DCMAKE_BUILD_TYPE=Release
```

7. Recompiler en mode `Release` et utiliser le mode verbose de `Make` :

```
desroziers> make clean
desroziers> make VERBOSE=1
```

8. Que constatez-vous en rapport avec la question 2. ?

2 Les bases du Machine Learning

Dans cette partie, nous allons découvrir et implémenter un algorithme de *Machine Learning* sur un problème simple de régression. Pour plus d'information, je conseille l'excellent cours d'Andrew Ng, fondateur du site `coursera` (voir `https://www.coursera.org/learn/machine-learning/`), plateforme qui dispense de nombreux MOOCs de qualité sur cette thématique.

Le contexte général du *Machine Learning* est d'appliquer une méthode dite d'*apprentissage* sur des *données* afin de *prédire* des *informations* utiles ou de déterminer des corrélations entre les données. Pour donner du sens à ceci, nous allons utiliser des données réelles (mais d'un volume réduit). Il existe de nombreux sites regroupant et ouvrant des données, que ce soit gouvernemental (en France, voir `https://www.data.gouv.fr/fr/datasets/`) ou pour la recherche en *Machine Learning* (voir par exemple `https://www.kaggle.com/datasets`). Pour cet exercice, on dispose d'un lot de données statistiques issues du secteur de la santé, c'est-à-dire le poids et taille d'un ensemble de 50 femmes et 50 hommes. Le but dans la suite est d'essayer de prédire le poids d'une personne en fonction de sa taille en utilisant ces données statistiques. En réalité, ces données sont très largement insuffisantes et incomplètes, il conviendrait d'étudier les nombreuses corrélations possibles avec par exemple l'age, le mode de vie ou les origines sur un gros volume de données issues d'une population représentative. Toutefois, la méthode proposée ici reste générale.

2.1 Régression linéaire à une variable

2.1.1 La méthode utilisée

On suppose dans cette partie que l'on dispose d'un ensemble de m couples $(t_i,\ p_i)_{i=1\dots m}$ où t_i est la taille du i^{me} individu (en cm) et p_i son poids (en kg). La méthode proposée ici repose sur l'hypothèse notée h_θ qu'il y ait une relation linéaire entre ces deux caractéristiques. Ainsi, on suppose alors que :

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x,$$

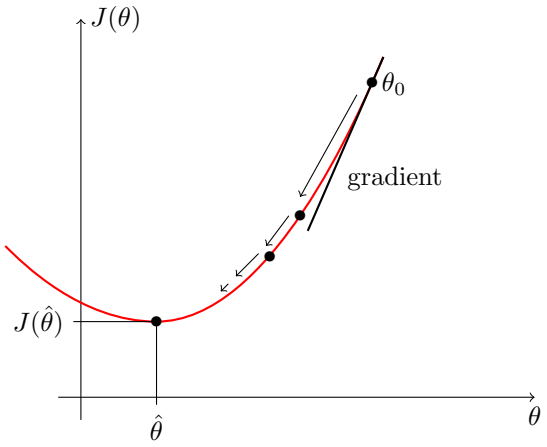
où $x \in \mathbb{R}$ ici représente la taille, $h_\theta(x)$ est notre hypothèse donnant le poids en fonction de la taille et $\hat{y} \in \mathbb{R}$ est une prédiction de poids. L'ensemble des prédictions est noté $Y \in \mathbb{R}^m$ et l'ensemble des caractéristiques (ou *features*) servant à la prédiction est noté $X \in \mathbb{R}^m$. Dans notre cas, $Y \equiv p$ et $X \equiv t$. Le vecteur de paramètres θ est à deux composantes, i.e. $\theta = \{\theta_0, \theta_1\} \in \mathbb{R}^2$ et c'est l'inconnue de notre problème qu'il nous faut découvrir. Une fois ce vecteur connu, il est alors possible d'utiliser l'hypothèse h_θ en prédiction. L'idée est de déterminer le vecteur θ de manière à ce que l'écart entre y_i et notre hypothèse $h_\theta(x_i)$ soit le plus petit possible pour l'ensemble des données.

Pour cela, nous utilisons une fonction dite *de coût* permettant pour un vecteur θ donné de mesurer l'écart entre l'hypothèse et les données. A moins d'avoir une hypothèse parfaitement adaptée aux données, cet écart n'est jamais nul mais on cherche à le minimiser. Le fonction de coût considérée est la suivante :

$$J(\theta) = \frac{1}{2m} \sum_{i=1\dots m} (h_\theta(x_i) - y_i)^2.$$

Ainsi, notre but est de trouver les meilleures valeurs de θ_0 et θ_1 minimisant la fonction $J(\theta)$. On note ce minimum $\hat{\theta}$, on suppose qu'il existe et qu'il est atteignable. Pour cela, on utilise une méthode itérative

de *descente de gradient* (voir `https://fr.wikipedia.org/wiki/Algorithme_du_gradient`). L'idée de cette méthode est de minimiser en suivant les *gradients* de la fonction. C'est un algorithme massivement utilisé en *Machine Learning* car il est peu coûteux et dispose de bonnes propriétés pour cette discipline. On parle alors d'*apprentissage* car d'une manière imagée, on apprend à partir des données pour fixer les paramètres afin de prédire efficacement.



L'algorithme de descente considéré consiste en une succession de corrections opérées de manière itérative sur le paramètre θ :

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i),$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}x_i(h_\theta(x_i) - y_i).$$

où on note α un paramètre dit d'*apprentissage*. Autrement dit, l'algorithme en pseudo-code est le suivant :

Algorithm 1 Calcul θ minimisant $J(\theta)$

Require: θ_0, θ_1, α
  $c = \frac{\alpha}{m}$
  for $k = 1...$ **do**
    $U_0 = 0$
    $U_1 = 0$
    for $i = 1...m$ **do**
      $h = h_\theta(x_i) - y_i$
      $U_0 = U_0 + ch$
      $U_1 = U_1 + chx_i$
    end for
    $\theta_0 = \theta_0 - U_0$
    $\theta_1 = \theta_1 - U_1$
  end for

2.1.2 Travail demandé

La commande `tree` appliquée au répertoire `ml` donne le résultat suivant :

```
desroziers> tree ml
ml
|-- CMakeLists.txt
|-- ex1f.cpp
|-- ex1m.cpp
|-- ex2.cpp

0 directories, 4 files
```

Votre travail dans cette partie va consister à compléter les quatre fichiers. Le but est d'utiliser au mieux les fonctions et les tableaux du `C++`. Nous améliorerons nos implémentations au fil des cours.

1. Modifier le fichier `CMakeLists.txt` pour compiler les trois fichiers `C++`. Précisons que le fichier est préalablement rempli pour permettre une compilation au standard `C++14`.
2. On considère ici le fichier `ex1m.cpp`. Ce fichier contient une variable `data` contenant les tailles et poids d'un échantillon de 50 hommes. Ecrire la conversion du tableau `data` en deux tableaux de 50 éléments `x` et `y`. Attention, `data` est un tableau monodimensionnel.
3. Implémenter la fonction `hypothesis` décrivant h_θ. Pour vous aider à vérifier votre implémentation, vous pouvez tout d'abord calculer à la main le résultat avec $\theta = \{1, 2\}$ et $x = 3$. Vérifier votre implémentation avec les valeurs suivantes :
   * $\theta_1 = \{1.0, 2.0\}$ et $x[11]$,
   * $\theta_2 = \{2.0, 2.5\}$ et $x[22]$,
   * $\theta_3 = \{0.5, 2.9\}$ et $x[33]$,
   * $\theta_4 = \{4.9, 2.2\}$ et $x[44]$.

```
verification hypothesis(x[11],theta1) = 364.936
verification hypothesis(x[22],theta2) = 422.048
verification hypothesis(x[33],theta3) = 510.082
verification hypothesis(x[44],theta4) = 382.147
```

4. Implémenter la fonction `computeCost` décrivant $J(\theta)$. Vérifier votre implémentation avec les tableaux `x` et `y` et les valeurs θ précédentes.

```
verification computeCost(x,y,theta1) = 35816.5
verification computeCost(x,y,theta2) = 63525.3
verification computeCost(x,y,theta3) = 90392.2
verification computeCost(x,y,theta4) = 47020.1
```

5. Implémenter la fonction `gradientDescent` décrivant l'algorithme de minimisation de $J(\theta)$. Vérifier votre implémentation avec les tableaux `x` et `y`, le paramètre $\alpha = 0.00001$ et le nombre d'itération du gradient $n = 20$.

```
paramètre theta obtenu : [0.00270733,0.482058]
```

6. Pour vérifier que le processus de minimisation soit correct, on peut effectuer une sortie de la fonction de coût $J(\theta)$ à chaque itération l'algorithme de descente de gradient. Modifier vos implémentations pour effectuer cette vérification et générer un fichier `costm.txt`. Pour visualiser, on utilise l'outil `gnuplot` (voir `http://www.gnuplot.info/`) qui permet de visualiser des courbes, nuages de points et bien plus. `gnuplot` est un éditeur en ligne de commande. Ouvrez l'éditeur `gnuplot` et afficher le fichier `costm.txt`. Que conclure ?

```
desroziers> gnuplot
gnuplot> plot "costm.txt"
```

7. Utiliser le paramètre θ obtenu pour prédire :
   * le poids d'un homme de 168 cm,
   * le poids d'un homme de 195 cm.

```
Pour un homme d'une taille de 168cm, on prédit un poids de 80.9884kg
Pour un homme d'une taille de 195cm, on prédit un poids de 94.004kg
```

8. Un fichier `datam.txt` contenant les données est généré. Ouvrez l'éditeur `gnuplot` et afficher les données ainsi que l'hypothèse.Que pensez-vous des résultats ?

```
desroziers> gnuplot
gnuplot> plot "datam.txt", 0.00270733+0.48205*x
```

9. On s'intéresse maintenant au fichier `ex1f.cpp`. Ce fichier contient une variable `data` contenant les tailles et poids d'un échantillon de 50 femmes. De la même façon que précédemment, écrire un algorithme de minimisation permettant de prédire le poids d'une femme à partir de sa taille. La convergence de la méthode pourra notamment être enregistrée dans le fichier `costf.txt`. On pourra également vérifier les étapes 3., 4. et 5. avec les mêmes données :

```
verification hypothesis(x[11],theta1) = 314.938
verification hypothesis(x[22],theta2) = 423.308
verification hypothesis(x[33],theta3) = 444.014
verification hypothesis(x[44],theta4) = 359.525
verification computeCost(x,y,theta1) = 34765.2
verification computeCost(x,y,theta2) = 59626.6
verification computeCost(x,y,theta3) = 83376.2
verification computeCost(x,y,theta4) = 44952.3
paramètre theta obtenu : [0.00225811,0.37142]
```

10. Utiliser le paramètre θ obtenu pour prédire :
    * le poids d'une femme de 150 cm,
    * le poids d'une femme de 175 cm.

```
Pour une femme d'une taille de 150cm, on prédit un poids de 55.7153kg
Pour une femme d'une taille de 175cm, on prédit un poids de 65.0008kg
```

11. Ouvrez l'éditeur et afficher l'ensemble des données de ce problème ainsi que les hypothèses. Que pensez-vous des résultats ?

```
desroziers> gnuplot
gnuplot> plot "dataf.txt", 0.00225811+0.37142*x, "datam.txt", 0.00270733+0.482058*x
```

12. Vous paraît-il possible de factoriser du code entre les fichiers `ex1m.cpp` et `ex1f.cpp` ? Comment faire ?

2.2 Régression linéaire à plusieurs variables

2.2.1 La méthode utilisée

Dans cette partie, on va construire un algorithme d'apprentissage permettant de mettre en relation un nombre quelconque de caractéristiques. Dans notre cas précis, on va ici considérer un ensemble de m triplets $(g_i,\ t_i,\ p_i)_{i=1...m}$ où g_i est le sexe du i^{me} individu (booléen `0` ou `1`), t_i sa taille (en cm) et p_i son poids (en kg). Ainsi, nous recherchons le vecteur paramètre $\theta = \{\theta_0, \theta_1, \theta_2\}$ en faisant l'hypothèse h_θ suivante :

$$\hat{y} = h_\theta(u, v) = \theta_0 + \theta_1 u + \theta_1 v,$$

où $u \in \mathbb{R}$ représente l'indicateur du sexe, $v \in \mathbb{R}$ la taille, $h_\theta(u, v)$ la fonction donnant le poids en fonction du sexe et de la taille et $\hat{y} \in \mathbb{R}$ une prédiction de poids.

Pour être général, on peut reformuler l'hypothèse de la manière suivante :

$$\hat{y} = h_\theta(x) = \sum_i \theta_i x_i = \theta^T x,$$

où $x \in \mathbb{R}^3$ est le vecteur $\{1, u, v\}$. Ainsi, l'ensemble des prédictions est noté $Y \in \mathbb{R}^m$ et l'ensemble des caractéristiques servant à la prédiction est noté $X \in \mathbb{R}^{3\times m}$. De plus, on a $Y \equiv p$ et $X \equiv \{1, g, t\}$. L'algorithme d'apprentissage de la partie précédente s'applique de manière générique en utilisant la nouvelle formulation de l'hypothèse.

Les caractéristiques de taille et l'indicateur de sexe ont des intervalles de valeurs très différents. Ce phénomène peut ralentir la convergence de l'algorithme. Il est d'usage de *normaliser* les données de X, c'est-à-dire ramener toutes les caractéristiques sur l'intervalle $[0, 1]$. Pour cela, on définit la *moyenne* d'une caractéristique $x \in \mathbb{R}^m$ comme :

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i$$

et son *écart type* :

$$\sigma = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (x_i - \mu)^2}.$$

On note $\overline{x}$ une caractéristique normalisée telle que $\overline{x} = \frac{x-\mu}{\sigma}$. Ainsi, on peut former les données normalisées $\overline{X}$ telles que $\overline{X} \equiv \{1, \overline{g}, \overline{t}\}$. Dans notre cas, seule la taille est à normalisée car l'indicateur de sexe a des valeurs 0 ou 1. La prédiction doit nécessairement tenir compte de la normalisation, i.e. pour une taille t et un indicateur de sexe g, on a :

$$\hat{y} = \theta_0 + \theta_1 g + \theta_2 \frac{t - \mu}{\sigma}.$$

2.2.2 Travail demandé

Un compte-rendu **individuel** de cette partie devra être produit. Les différents éléments du compte-rendu (code source, rapport au format PDF) sont personnels, doivent être soignés et devront être envoyés à l'adresse `sylvain.desroziers@ifpen.fr`. Il est autorisé de travailler à plusieurs mais les **collègues de travail doivent être cités** dans les rapports.

1. On s'intéresse maintenant au fichier `ex2.cpp`. Ce fichier contient une variable `data` contenant les tailles et poids d'un échantillon de 50 hommes et 50 femmes. Ecrire la conversion du tableau `data` en deux tableaux de 50 éléments `X` et `y`. Attention, une ligne du tableau `X` contient 3 éléments dont le premier vaut toujours la valeur `1`. Les deux autres éléments sont les caractéristiques sexe et taille des données.
2. Implémenter une fonction calculant l'hypothèse h_θ. Vérifier votre implémentation avec les valeurs suivantes :
   * $\theta_1 = \{1.0, 2.0, 3.0\}$ et $x[11]$,
   * $\theta_2 = \{2.0, 2.5, 3.5\}$ et $x[33]$,
   * $\theta_3 = \{0.5, 2.9, 3.9\}$ et $x[55]$,
   * $\theta_4 = \{4.9, 2.2, 1.1\}$ et $x[77]$.
3. Implémenter une fonction calculant la fonction de coût $J(\theta)$. Vérifier votre implémentation avec les tableaux `X` et `y` et les valeurs θ précédentes.
4. Implémenter une fonction calculant la moyenne d'une caractéristique de `X`. Vérifier votre implémentation avec le tableau `X` pour la deuxième caractéristique (i.e. la taille).
5. Implémenter une fonction calculant l'écart type d'une caractéristique de `X`. Vérifier votre implémentation avec le tableau `X` pour la deuxième caractéristique et la moyenne précédemment calculée.
6. Implémenter une fonction appliquant la normalisation d'une caractéristique de `X`. Après l'appel de la fonction, le tableau `X` a ses valeurs normalisées.
7. Implémenter la fonction de minimisation de $J(\theta)$. Vérifier votre implémentation avec les tableaux `X` après normalisation de la deuxième caractéristique et `y`, le paramètre $\alpha = 0.1$ et le nombre d'itération du gradient $n = 50$. Votre implémentation devra générer un fichier `cost.txt` permettant de valider la convergence.
8. Utiliser le paramètre θ obtenu pour prédire :
   * le poids d'une femme de 150 cm,
   * le poids d'une femme de 175 cm,
   * le poids d'un homme de 168 cm,
   * le poids d'un homme de 195 cm.
9. Vous paraît-il possible de factoriser du code entre les fichiers `ex1m.cpp`, `ex1f.cpp` et `ex2.cpp` ? Comment faire ?