TP 1 - Régression linéaire

Rendu : Vous êtes encouragés à échanger et à discuter vos résultats au cours du TP. Néanmoins, le compte-rendu est un travail **individuel**.
Il vous est demandé de rendre un **notebook ipython** où vous répondrez aux questions dans l'ordre. Vous inclurez des cellules de texte pour commenter vos résultats, comme il vous l'est demandé tout au long de ce TP. Une attention particulière devra être portée à la clarté des figures (axes, titre...).
Si (et seulement si) vous n'avez pas pu installer Jupyter, vous pouvez rendre un compte-rendu classique incluant un fichier PDF avec vos réponses aux questions d'une part, et un fichier avec votre code (clair et commenté) d'autre part. Vous définirez une fonction Python par question (qui pourra faire appel à d'autres fonctions), comme dans le bout de code suivant.

```
def question11():
    X,Y = gen_linear()
    plt.figure()
    plt.scatter(X,Y)
```

Ce TP est destiné à l'étude de la régression linéaire. Le problème de régression consiste à prédire une valeur réelle y à partir d'un certain nombre d'entrées réelles : $(x_1, x_2, \ldots, x_d)$. On dispose pour cela d'une base d'apprentissage constitué d'*exemples* $\mathbf{x}^i$ décrit généralement dans $\mathbb{R}^d$ et de leur label/étiquette y^i dans $\mathbb{R}$. Formellement, il s'agit de trouver une fonction $f : \mathbb{R}^d \to \mathbb{R}$ *en accord* avec la base d'apprentissage.
Afin d'estimer cet accord, une fonction de coût $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$ est utilisée pour rendre compte de l'erreur entre la valeur prédite (généralement notée $\hat{y} = f(\mathbf{x})$) et la valeur attendue y. Le problème d'apprentissage consiste alors à trouver une fonction qui minimise le risque, i.e. l'espérance de la fonction de coût sur la distribution sous-jacente des données : $f^* = argmin_f\ R(f) = argmin_f \mathbb{E}_{\mathbf{x},y}\left(\ell\left(f(\mathbf{x}), y\right)\right)$.
La *minimisation du risque empirique* consiste à remplacer l'espérance par une estimation sur les données disponibles, c'est-à-dire la base d'apprentissage $\hat{f} = argmin_f\ \hat{R}(f) = argmin_f \sum_{i=1}^{n} \ell(f(\mathbf{x}^i), y^i)$.
La fonction de coût la plus utilisée en régression est le coût au sens des moindres carrés : $\ell(\hat{y}, y) = (\hat{y} - y)^2$. C'est cette fonction que nous utiliserons dans ce TP.

Conventions : dans toute la suite, nous représenterons les données sous la forme d'une matrice $X \in \mathbb{R}^{n \times d}$, chaque ligne de X étant un exemple et chaque colonne une variable ; les labels par une matrice colonne $Y \in \mathbb{R}^{n \times 1}$. Par ailleurs, nous utiliserons les abréviations usuelles pour les modules Python : `import matplotlib.pyplot as plt` et `import numpy as np`.
Attention :

- `numpy` fait une très grande différence entre un vecteur et une matrice ! Un vecteur est tel que la fonction `shape` renvoie un tuple de longueur 1, une matrice un tuple de longueur 2. Les opérations matricielles `np.dot` (produit matriciel ou scalaire) et `*` (produit terme à terme **ou** matriciel) dépendent fortement de la nature de l'objet, matrice ou vecteur. Vous aurez beaucoup d'erreurs au début provenant de cette confusion ...
- Vous utiliserez souvent la fonction `np.reshape` pour vous assurer d'avoir un vecteur ou une matrice d'une certaine dimension. Vous pouvez passer un paramètre à -1 dans une dimension afin que la fonction calcule automatiquement la dimension voulue. Par exemple, `X.reshape(-1,1)` permet de transformer en une matrice colonne la matrice `X`.
- Évitez à tout prix de faire des boucles en Python quand vous pouvez utiliser des opérations matricielles ou mathématiques ! Python est un langage interprété et de ce fait (pourquoi ?) les boucles sont très lentes ! N'hésitez pas à utiliser les opérateurs `np.mean`, `np.dot`, `np.where`, `np.hstack`, `np.vstack`, ...

1 Données artificielles et réelles

Nous utiliserons souvent dans cette série de TPs des données artificielles en 1D ou 2D afin de visualiser les résultats. Pour cela, prenez bien en main les fonctions du module `numpy.random` qui permettent de faire de la génération de données aléatoires.

Q 1.1 Génération aléatoire

Q 1.1.1 Programmer la fonction `gen_linear(a,b,eps,nbex)` qui permet d'engendrer `nbex` points (x^i, y^i) i.i.d. suivant $y = ax + b + \epsilon$, avec :

- ϵ un bruit gaussien de variance `eps`.
- x tiré selon une loi uniforme sur le segment $[-5, 5]$.

La fonction devra retourner deux matrices X de taille `(nbex,1)` et Y de taille `(nbex,1)`.

Q 1.1.2 Visualiser à l'aide de `plt.scatter` les points engendrés.

Q 1.1.3 Généraliser votre fonction précédente pour un vecteur d'entrée à d variables, en passant à la place de `a,b` une liste de coefficients. Dans la suite, on prendra $a = [1, 1, 1]$ et $b = 5$

Q 1.2 Données réelles : boston housing (prix des logements à Boston en fonction de critères socio-économiques).

Q 1.2.1 Télécharger le fichier `housing.csv`. Charger le avec le bout de code ci-dessous. Observer les moyennes et variances de chaque dimension.

```
def read_file(fn):
    with open(fn) as f:
        names =f.readline()
        X=np.array([[float(x) for x in l.strip().split(" ")] for l in f.readlines()])
    return X[:,:-1],X[:,-1].reshape(-1)
```

Q 1.2.2 Pour quelques paires de variables (i, j), représentez en 2D le nuage de points obtenus en restreignant les données d'entrée à ces variables. Vous pourrez utiliser le bout de code suivant pour afficher plusieurs graphiques dans une même figure.

```
plt.figure()
plt.subplot(5,5,1) #plt.subplot(rows,cols,idx)
plt.scatter(X1,Y1)
plt.subplot(5,5,2)
plt.scatter(X2,Y2)
```

Des variables vous semblent-elles corrélées ?

2 Régression linéaire

La régression linéaire suppose une relation linéaire entre la valeur à prédire - l'étiquette - et les différentes dimensions. La forme générique est donc $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$ avec $\mathbf{w} \in R^{d+1}$ le vecteur de paramètres.
Vous aurez besoin en particulier des fonctions `np.dot(), np.transpose()` (ou `X.dot(), X.T`), `np.linalg.pinv(), np.mean()`.

Q 2.1 (Préambule) Exprimer $f_{\mathbf{w}}(\mathbf{x})$ par un produit scalaire, puis $Y = f_{\mathbf{w}}(X)$ le résultat de $f_{\mathbf{w}}$ sur toute la base par un produit matriciel. Donner sous forme matricielle le calcul de l'erreur des

moindres carrés pour une base d'apprentissage X, Y. Retrouver la solution analytique de l'estimateur des moindres carrés.

Q 2.1.1 Programmer la fonction `predict(w,X)` qui permet d'obtenir le vecteur résultat de $f_{\mathbf{w}}(X)$.

Q 2.1.2 Donner la fonction `mse(yhat,y)` qui renvoie l'erreur au sens des moindres carrés entre le vecteur `y` et `yhat`.

Q 2.1.3 Programmer la fonction `regress(X,Y)` qui renvoie la solution $\mathbf{w}$ sous forme d'un vecteur.

Q 2.2 Tester sur les données artificielles. Faire varier le nombre de données utilisées et le bruit. Comment évolue l'erreur des moindres carrés ? L'écart quadratique entre les poids estimés et les vrais coefficients ? Introduire une ou plusieurs dimensions aléatoires. Que remarquez vous pour les résultats ? Pour les poids de ces dimensions ? Tracer toutes les courbes correspondantes.

Q 2.3 Tester sur les données réelles.

Q 2.3.1 Pouvez-vous donner une interprétation aux valeurs des poids des différentes variables ?

Q 2.3.2 Multiplier par 10 une des dimensions. Que se passe-t-il pour l'erreur ? Pour le poids associé ? Cela modifie-t-il votre interprétation de la question précédente ? Quel pré-traitement sur les données permet de donner aux poids des variables une interprétation plus fiable ? Tracer la courbe qui donne l'erreur des moindres carrés en fonction du nombre de données utilisées, en appliquant ce pré-traitement.

Q 2.3.3 Avez-vous confiance en l'estimation de l'erreur des moindres carrés ? Partager les données initiales en deux sous-ensembles, un sous-ensemble d'apprentissage et un de test. Pour un centrage et une normalisation donnée du sous-ensemble d'apprentissage, comment faut-il traiter les données de test pour évaluer correctement le modèle linéaire sur celles-ci ? Comparer l'erreur des moindres carrés sur les deux sous-ensembles pour différentes tailles des sous-ensembles et tracer les courbes correspondantes.

3 Ridge Régression

Q 3.1 Quel problème peut arriver dans la résolution analytique de la régression linéaire ? Dans quel cas ? Cela arrive-t-il souvent ? Afin de pallier ce problème, la ridge régression introduit un terme supplémentaire dans la fonction de coût en y ajoutant $\lambda\|\mathbf{w}\|^2$, la norme 2 au carré du vecteur de paramètres. Ce terme peut également s'interpréter comme une pénalisation sur les poids du vecteur solution trouvé - une régularisation. Exprimer de nouveau la solution analytique sous forme matricielle. Programmer la fonction `ridge_regress(X,Y,lmb)`.

Q 3.2 Expérimenter sur les données artificielles : tracer les différentes erreurs en fonction du paramètre λ. On prendra λ sur une grille logarithmique en veillant à couvrir les régimes intéressants.

Q 3.3 Expérimenter de manière analogue sur les données réelles. Les résultats sont-ils différents des questions précédentes ?

4 LASSO

Le Lasso remplace la norme l_2 au carré par une norme l_1. On résout dans ce cas le problème suivant :

$$\hat{f} = argmin_w \frac{1}{2n} \sum_{i=1}^{n} \|Xw - Y\|_2^2 + \lambda \cdot \|w\|_1$$

L'algorithme est implémenté dans scikit-learn dans le module `sklearn.linear_model.Lasso`. Plus de détails sur l'implémentation sont disponibles dans la documentation de scikit-learn.
On travaille dans cette partie sur les données réelles.

Q 4.1 Tracer, sur la même figure, l'évolution de chaque composante du vecteur w en fonction de λ. Faites de même pour la ridge régression. Que remarquez vous ?

Q 4.2 Comparer la performance du Lasso et celle de la ridge régression sur les données de test.

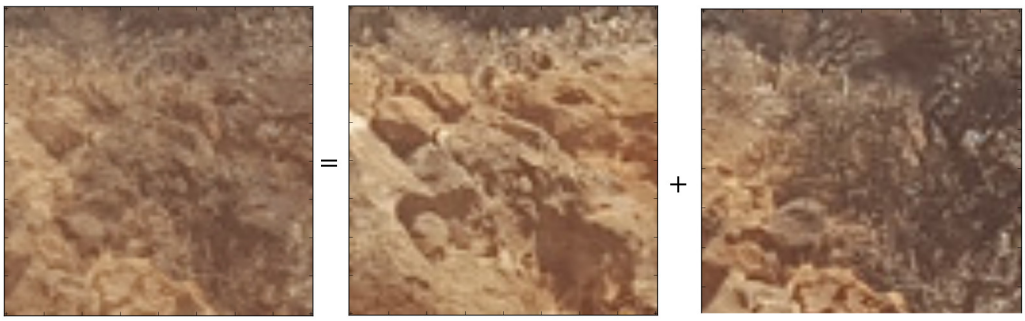
Dans la question suivante, on considère une méthode hybride qui, pour une valeur de λ fixée, utilise le Lasso pour sélectionner des variables (celles qui obtiennent un coefficient non nul) et résout une régression ordinaire sur les variables sélectionnées.

Q 4.3 (bonus) Comparer pour différentes valeur de λ le Lasso avec la méthode hybride. Tracer sur une même figure les performances des deux méthodes en fonction du nombre de variable sélectionnées. Que constatez-vous ? Pouvez-vous l'expliquer ?

5 Bonus : Inpainting

L'*inpainting* en image s'attache à la reconstruction d'images détériorées ou au remplissage de parties manquantes (éliminer une personne ou un objet d'une image par exemple). Cette partie est consacrée à l'implémentation d'une technique d'inpainting présentée dans [1] utilisant le Lasso et sa capacité à trouver une solution parcimonieuse en termes de poids. Ces travaux sont inspirés des recherches en *Apprentissage de dictionnaire et représentation parcimonieuse* pour les signaux [2]. L'idée principale est de considérer qu'un signal complexe peut être décomposé comme une somme pondérée de signaux élémentaires, comme c'est le cas également dans l'ACP. les signaux élémentaires - nommés atomes - peuvent être cependant dans ce cadre fortement redondants. La difficulté est donc double : réussir à construire un dictionnaire d'atomes - les signaux élémentaires - et trouver un algorithme de décomposition/recomposition d'un signal à partir des atomes du dictionnaire, i.e. trouver des poids parcimonieux sur chaque atome tels que la combinaison linéaire des atomes permettent d'approcher le signal d'origine. Dans la suite, nous étudions une solution pour la deuxième question à l'aide de l'algorithme du Lasso.

Formalisation Pour une image de dimension 2, nous noterons $p_i \in [0,1]^3$ le i-ème pixel de l'image exprimée sur 3 canaux [1]. Un patch de taille h centré au pixel p_i correspond a un petit carré de pixels dans l'image de longueur h donc le centre est le pixel p_i. Nous le noterons par la suite Ψ^{p_i} (la taille h sera constante). Ce patch correspond à un tenseur (matrice 3d ici) de taille $h \times h \times 3$ que l'on peut voir sans perte de généralité comme un vecteur colonne de taille $3h^2$; à une image de taille $(width, height)$, il est possible d'associer l'ensemble des patchs $\mathbf{\Psi}$. L'hypothèse fondamentale est qu'une image a une cohérence spatiale et de texture : un patch Ψ^p appartenant à une région cohérente en termes de texture doit pouvoir être reconstruit à partir des patchs environnants : $\Psi^p = \sum_{\Psi^{p_k} \in \mathbf{\Psi} \setminus \Psi^p} w_k \Psi^{p_k}$.



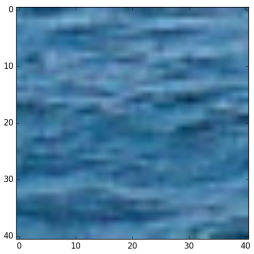
Dans la suite, nous allons considérer qu'une grande partie de l'image n'a pas de pixels manquant. Les patchs issus de cette partie de l'image constitueront notre dictionnaire d'atomes $\mathbf{\Psi}$ sur lesquels porteront nos combinaisons linéaires afin de reconstituer les parties manquantes.

1. Les 3 canaux sont usuellement le pourcentage de rouge, vert et bleu (rgb) ou sous format teinte, saturation, luminosité (hsv), plus proche de notre perception visuelle

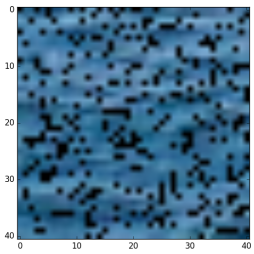
Supposons qu'un certain nombre des pixels d'un patch soient manquants : soit I l'ensemble des entrées manquantes correspondantes dans le vecteur Ψ^p et I^c son complémentaire. Nous noterons $\Psi^p_{I^c}$ l'ensemble des pixels exprimés du patch. La reconstruction consiste à faire l'hypothèse que si une combinaison linéaire est performante pour approximer $\Psi^p_{I^c}$, elle sera capable de généraliser aux valeurs manquantes Ψ^p_I. Le problème d'optimisation consiste dans ce cas à trouver le vecteur $\hat{\mathbf{w}}$ tel que :

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \|\Psi^p_{I^c} - \sum_{\Psi^k \in \mathbf{\Psi}} w_k \Psi^k_{I^c}\|_2^2 + \lambda \|\mathbf{w}\|_1$$

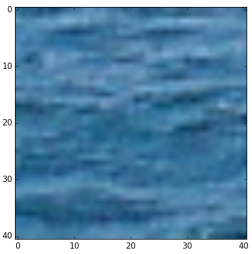
Autrement dit, les pixels présents sont utilisés pour l'apprentissage du vecteur $\mathbf{w}$ de régression et les valeurs manquantes sont les valeurs à prédire.



Patch origine



Patch bruité



Patch débruité

Q 5.1 Télécharger le fichier `inpainting.py`. Il contient quelques fonctions utiles pour charger une image, extraire un patch, bruiter un patch, transformer un patch en vecteur et inversement, afficher un patch et construire un dictionnaire de patchs. Implémenter une fonction `denoise(patch,dic)` qui permet de débruiter le patch donné en entrée en fonction du dictionnaire `dic`. Faites varier le coefficient de régularisation. Que remarquez vous pour les valeurs faibles et grandes de λ ?

Q 5.2 Supposons maintenant que c'est toute une partie de l'image qui est manquante. En considérant des patchs centrés sur les pixels manquants, on commence par remplir en partant des bords puis en remplissant au fur et à mesure vers le centre de l'image. A votre avis, l'ordre de remplissage a-t-il une importance ? Implémenter une fonction qui permet de compléter l'image. Proposer des heuristiques pour remplir de manière intelligente (voir [3]).

Références

[1] Bin Shen and Wei Hu and Zhang, Yimin and Zhang, Yu-Jin, *Image Inpainting via Sparse Representation* Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '09)

[2] Julien Mairal *Sparse coding and Dictionnary Learning for Image Analysis* INRIA Visual Recognition and Machine Learning Summer School, 2010

[3] A. Criminisi, P. Perez, K. Toyama *Region Filling and Object Removal by Exemplar-Based Image Inpainting* IEEE Transaction on Image Processing (Vol 13-9), 2004