Apprentissage supervisé

Les sources et liens utiles sont sur le site du cours et/ou de l'auteur. Commencez par télécharger des fichiers utiles pour ce TP :
- le fichier source `tp_ML_supervise_source.py`
- un exemple de scripte d'utilisation `tp_ML_supervise_scripte.py`

- Découverte de Python -

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels
⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/1-Intro-Python.html
⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/2-Numpy.html
⋆⋆⋆ http://perso.telecom-paristech.fr/~gramfort/liesse_python/3-Scipy.html
⋆⋆⋆ http://scikit-learn.org/stable/index.html
⋆⋆ http://www.loria.fr/~rougier/teaching/matplotlib/matplotlib.html
⋆⋆ http://jrjohansson.github.io/

- Classification binaire -

Introduction

Définitions et notations

On rappelle que dans la cadre de la classification binaire supervisée on utilise les notations :
- $\mathcal{Y}$ l'ensemble des étiquettes (ou *labels* en anglais), communément $\mathcal{Y} = \{-1, 1\}$ dans le cas de la classification binaire,
- $\mathbf{x} = (x_1, \dots, x_p) \in \mathcal{X} \subset \mathbb{R}^p$ est un attribut (ou un *feature* en anglais),
- $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$ un ensemble d'apprentissage contenant n exemples et leurs étiquettes,
- Il existe un modèle probabiliste qui gouverne la génération de nos observations selon des variables aléatoires X et Y : $\forall i \in \{1, \dots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.
- On cherche à construire à partir de l'ensemble d'apprentissage $\mathcal{D}_n$ une fonction $\hat{f} : \mathcal{X} \mapsto \{-1, 1\}$ qui pour un point inconnu (*i.e.,* qui n'est pas présent dans l'ensemble d'apprentissage) prédit son étiquette : $\hat{f}(\mathbf{x})$.

Génération artificielle de données

Dans un but d'expérimentation, il est plus aisé de travailler sur des données engendrées artificiellement. On considère dans cette partie que les observations sont décrites en deux dimensions (afin de pouvoir les visualiser facilement).

1. Étudiez la fonction `rand_gauss(n,mu,sigma)` qui engendre n observations selon la loi normale multidimensionnelle de moyenne le vecteur `mu` et de matrice de covariance considérée diagonale de diagonale `sigma`. Étudiez ensuite les fonctions `rand_bi_gauss`, `rand_clown` et `rand_checkers`. Que renvoient ces fonctions ? À quoi correspond la dernière colonne ?
2. Sauvegardez quelques jeux de données afin de les utiliser dans la suite : pour chacun, il faudra sauver dans une variable les données sous forme d'un tableau à deux colonnes, et dans une autre les labels correspondants à chaque exemple.
3. Utilisez la fonction `plot_2d` afin d'afficher quelques jeux de données.

Extensions aux cas multi-classe

Dans le cas où la variable de sortie Y compte plus de deux modalités, il existe plusieurs façon d'étendre directement les méthodes du cas binaire.

"Un contre un". Dans le cas où l'on cherche à prédire un label pouvant prendre $K \geq 3$ modalités, on peut considérer toutes les paires de labels (k, l) possibles, $1 \leq k < l \leq K$ (il y en a C_K^2) et ajuster un classifieur $C_{k,l}(X)$ pour chacune d'entre elles. La prédiction correspond alors au label qui a gagné le plus de "duels".

"Un contre tous". Pour chaque modalité k, on apprend un classifieur permettant de discriminer entre les populations $Y = k$ et $Y \neq k$. À partir des estimations des probabilités a posteriori, on affecte le label estimé le plus probable.

Analyse Discriminante Linéaire

Le nom anglais est *Linear Discirmiant Analysis* (LDA) et il est utile pour trouver de l'aide pour la partie numérique.

Aspect théorique

On considère deux populations gaussiennes dans $\mathbb{R}^p$ ayant la **même** structure de covariance. On observe des points dans le mélange de ces deux populations.

Les lois conditionnelles de X sachant $Y = +1$ (respectivement $Y = -1$) sont des gaussiennes multivariées $\mathcal{N}_p(\mu_+, \Sigma)$ (respectivement $\mathcal{N}_p(\mu_-, \Sigma)$). On notera leur densités respectives f_+ et f_-. Les vecteurs μ_+ et μ_- sont dans $\mathbb{R}^p$ et la matrice Σ est (symétrique) de taille $p \times p$. On note également $\pi_+ = \mathbb{P}\{Y = +1\}$.

On rappelle que la densité p-dimensionnelle de la loi $\mathcal{N}_p(\mu, \Sigma)$ est donnée par :

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{\det(\Sigma)}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right\} .$$

et que la matrice de covariance d'un vecteur aléatoire X est définie par $\Sigma = \mathbb{E}\big[(X - \mathbb{E}(X))(X - \mathbb{E}(X))^\top\big]$.

1. En utilisant la formule de Bayes donner la formule des probabilités a posteriori : $\mathbb{P}\{Y = +1 \mid X = \mathbf{x}\}$, $\mathbb{P}\{Y = -1 \mid X = \mathbf{x}\}$, comme fonctions de f_+, f_- et π_+.
2. Exprimer le log-ratio des deux classes :

$$\log\left(\frac{\mathbb{P}\{Y = +1 \mid X = \mathbf{x}\}}{\mathbb{P}\{Y = -1 \mid X = \mathbf{x}\}}\right)$$

   en fonction de π_+, μ_+, μ_- et Σ.
3. On dispose à présent d'un échantillon de ce mélange et on suppose que π_+, μ_+, μ_- et Σ sont des paramètres inconnus. On suppose que l'échantillon considéré contient n observations notées $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ et que $\sum_{i=1}^n \mathbb{1}\{y_i = +1\} = m$. En utilisant la méthode des moments utilisée en estimation paramétrique, proposer des estimateurs $\widehat{\pi}_+$, $\widehat{\mu}_+$, $\widehat{\mu}_-$ et $\widehat{\Sigma}$ des paramètres.
4. Justifier le choix du classifieur suivant :

$$\begin{cases} 1 & \text{si } \mathbf{x}^\top \widehat{\Sigma}^{-1}(\widehat{\mu}_+ - \widehat{\mu}_-) > \frac{1}{2}\widehat{\mu}_+^\top \widehat{\Sigma}^{-1}\widehat{\mu}_+ - \frac{1}{2}\widehat{\mu}_-^\top \widehat{\Sigma}^{-1}\widehat{\mu}_- + \log(1 - m/n) - \log(m/n) \ , \\ -1 & \text{sinon} \end{cases}$$

5. Que se passe-t-il lorsque les matrices de covariance ne sont pas égales ? Quelle forme prend la frontière ?
6. Comment généraliser l'analyse discriminante linéaire au cas multi-classes ?

Mise en oeuvre

Il s'agit ici d'appliquer concrètement la méthode ci-dessus sur des observations simulées puis à une base de données réelles. Dans ce dernier cas, on scindera aléatoirement les données en deux échantillons : un échantillon d'apprentissage (70% des données environ) à partir duquel les paramètres régissant le score seront estimés et un échantillon test (les 30% restant) sur lequel on évaluera la performance de la règle de score précédemment apprise.

Le paquet `sklearn` met à disposition un grand nombre d'algorithmes usuels de l'apprentissage statistique. Pour commencer, importez le paquet `sklearn.lda` qui contient en particulier la classe `LDA` qui nous servira d'exemple dans la suite.

```
from sklearn.lda import LDA
```

Pour utiliser ces algorithmes, il est nécessaire de procéder par étape en Python. Nous allons voir en détail le processus pour l'analyse discriminante linéaire tout d'abord.

7. Créer un modèle `LDA` :

```
my_lda = LDA()
```

   Vous pouvez paramétrer le modèle lors de sa création (en passant des arguments) ou bien par la suite directement par l'intermédiaire de la variable.

8. Afin d'apprendre le modèle sur une matrice de données `dataX` et de labels `dataY`, on utilise `fit`.

```
my_lda.fit(dataX,dataY)
```

9. Appliquer l'analyse discriminante linéaire sur les données du mélange de gaussiennes générées par `rand_bi_gauss`. Estimer son erreur de prédiction au moyen de l'échantillon test (comparer l'estimation à l'"erreur d'apprentissage"). Tracer la frontière.
10. Appliquer la classification par régression logistique sur les données non gaussiennes `rand_clown` et `rand_checkers`.

Régression logistique

Importer le paquet `sklearn.linear_model` qui contient en particulier la classe `LogisticRegression` qui nous servira d'exemple dans la suite.

```
from sklearn import linear_model
```

Pour utiliser ces algorithmes, il est nécessaire de procéder par étape en Python. Nous allons voir en détail le processus pour la régression logistique.

1. Créer un modèle `LogisticRegression` :

```
my_log = linear_model.LogisticRegression()
```

   Vous pouvez paramétrer le modèle lors de sa création (en passant des arguments) ou bien par la suite directement par l'intermédiaire de la variable

2. Afin d'apprendre le modèle sur une matrice de données `dataX` et de labels `dataY`, on utilise `fit`

```
my_log.fit(dataX,dataY)
```

   On pourra s'inspirer de l'exemple suivant pour utiliser des variantes de régression : http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#example-linear-model-plot-ols-py
   http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html

3. À quoi correspond la variable `coef_` du modèle ? `intercep_` ?
4. Que vous retourne la fonction `predict` ? Et la fonction `score` ?
5. Utiliser la fonction `frontiere` pour visualiser la frontière de décision.
6. Appliquer la classification par régression logistique es données issues de la base ZIPCODE (modalités 2 et 3) disponibles sur le site http://www-stat.stanford.edu/ElemStatLearn.

Le perceptron

Un perceptron est un classifieur linéaire binaire qui projette chaque observation $\mathbf{x}$ dans $\mathbb{R}$. L'ensemble des frontières de décision considérées est alors l'ensemble des hyperplans (affine) de $\mathbb{R}^p$ définis pour un certain vecteur $\mathbf{w} = (w_0, w_1, \cdots, w_p) \in \mathbb{R}^{p+1}$ par

$$H_{\mathbf{w}} = \left\{ \mathbf{x} : \hat{f}_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i = 0 \right\}.$$

Pour classer une observation $\mathbf{x}$, il s'agit simplement de considérer la position de $\mathbf{x}$ par rapport à l'hyperplan $H_{\mathbf{w}}$. Cela revient à prédire comme étiquette de $\mathbf{x}$ la valeur de $\text{sign}\left(\hat{f}_{\mathbf{w}}(\mathbf{x})\right)$, où la fonction sign est définie par

$$\text{sign}(x) = \begin{cases} 1 & \text{si } x > 0, \\ -1 & \text{si } x < 0, \\ 0 & \text{sinon.} \end{cases}$$

L'objectif est de trouver l'hyperplan qui sépare "au mieux" les données selon leur étiquette. Le vecteur $\mathbf{w}$ est appelé vecteur de poids (et w_0 l'ordonnée à l'origine, ou *intercept* en anglais).

Perceptron simple

Nous supposerons dans cette partie introductive que nos données sont en deux dimensions ($p = 2$). Utilisez les données artificielles pour illustrer les questions suivantes.

1. A quoi correspond la frontière de décision du perceptron ? Trouvez (à la main) une bonne séparatrice. Quand est-ce que $\hat{f}_{\mathbf{w}}(\mathbf{x})$ est grand ? négatif ? positif ? Quelle signification géométrique ?
2. Codez une fonction `predict(x,w)` qui à partir d'une matrice $\mathbf{x}$ et d'un vecteur poids $\mathbf{w}$ renvoie le vecteur de prédiction $\hat{f}_{\mathbf{w}}(\mathbf{x})$. Codez de même une fonction `predict_class(x,w)` qui renvoie le vecteur d'étiquettes prédites $\text{sign}\left(\hat{f}_{\mathbf{w}}(\mathbf{x})\right)$.

Fonction de coût

Afin de mesurer l'erreur sur l'ensemble d'un jeu de données $\mathcal{D}_n$ il est nécessaire de se fixer une fonction de perte $\ell : \mathcal{Y} \times \mathbb{R} \mapsto \mathbb{R}^+$ qui mesure le coût d'une erreur lors de la prédiction d'un exemple. Le coût $C_{\mathbf{w}} = \mathbb{E}\left[\ell(\hat{f}_{\mathbf{w}}(\mathbf{x}), y)\right]$ est l'espérance de la fonction de perte sur l'ensemble des données.

Trois fonctions de perte sont utilisées habituellement :

- le pourcentage d'erreur : $\text{ZeroOneloss}(y, \hat{f}_{\mathbf{w}}(\mathbf{x})) = |y - \text{sign}(\hat{f}_{\mathbf{w}}(\mathbf{x}))|/2$
- l'erreur quadratique : $\text{MSEloss}(y, \hat{f}_{\mathbf{w}}(\mathbf{x})) = (y - \hat{f}_{\mathbf{w}}(\mathbf{x}))^2 =$
- l'erreur *hinge* (*i.e.,* charnière en français) : $\text{HingeLoss}(y, \hat{f}_{\mathbf{w}}(\mathbf{x})) = \max(0, 1 - y \cdot \hat{f}_{\mathbf{w}}(\mathbf{x}))$

Cette partie à pour but d'étudier ces différentes fonction de pertes.

3. Codez ces trois fonctions (en bloc de préférence, de manière à ce que $\mathbf{x}$ soit la matrice de données, et y le vecteur des labels et qu'elles renvoient le vecteur des pertes).
4. Pour un $\mathbf{w}$ fixé, sur un exemple 2D, comment varient ces fonctions en fonction de $\hat{f}_{\mathbf{w}}(\mathbf{x})$? Et de $\mathbf{x}$? Quelle est l'interprétation géométrique ?
5. Comment observer graphiquement l'évolution de ces fonctions selon $\mathbf{w}$ pour une base d'exemples fixée en 2D (pensez à l'utilité de w_0) ? Tracer en 2D (grâce à la fonction fonction `frontiere` par exemple) le coût sur une base d'exemple en fonction de $\mathbf{w}$. Où se situe le vecteur

$$\mathbf{w}^* \in \underset{\mathbf{w} \in \mathbb{R}^{p+1}}{\arg\min} \left(\frac{1}{n} \sum_{i=1}^{n} \ell(\hat{f}_{\mathbf{w}}(\mathbf{x}_i), y_i) \right)$$

minimisant l'erreur empirique (correspondant à la séparatrice minimisant l'erreur) ? Quelle(s) propriété(s) remarquable(s) possèdent (ou non) ces fonctions ?

Apprentissage du perceptron - Algorithme de descente du gradient

Dans le cas général, il est bien sûr impossible de faire une recherche exhaustive de l'espace $\mathbb{R}^{p+1}$ où évolue $\mathbf{w}$ afin de trouver le minimum. On utilise pour cela l'algorithme de descente du gradient, une méthode usuelle et générale d'optimisation de fonction différentiable. La méthode est itérative : à chaque pas, le point courant est corrigé dans la direction du gradient, mais en sens opposé. L'algorithme converge dans le cas général vers un minimum local. Le minimum atteint est global en particulier pour les fonctions convexes.

L'algorithme est le suivant :

Algorithme 1 : Perceptron

Data : les observations et leurs étiquettes $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n\}$; le pas de gradient : ϵ ; le nombre maximal d'itérations : n_{iter} ;
Result : $\mathbf{w}$
initialiser (aléatoirement) $\mathbf{w}$; initialiser $j = 0$
while $j \leq n_{\text{iter}}$ **do**
  $\mathbf{w}_{\text{old}} \leftarrow \mathbf{w}$
  for $i = 1$ *to* n **do**
    $\mathbf{w} \leftarrow \mathbf{w}_{\text{old}} - \epsilon \nabla_{\mathbf{w}} \ell(\hat{f}_{\mathbf{w}_{\text{old}}}(\mathbf{x}_i), y_i)$
  $j \leftarrow j + 1$

6. Expérimentez sur différents jeux de données : utiliser soit les fonctions fournies, soit le paquet `sklearn`. On peut aussi utiliser à cet effet la fonction SGD (pour *Stochastique Gradient Descent*) dont une description est donnée sur la page : http://scikit-learn.org/stable/modules/sgd.html

```
from sklearn.linear_model import SGDClassifier
```

   Étudiez les performances au moins selon les points suivants : nombre d'itérations, le choix de la fonction de coût, la difficulté du problème. Observez vous des comportements étranges ? Quelles raisons ?
7. Une variante de l'algorithme, dite stochastique (le précédent est communément appelé *batch*, *i.e.*, lot ou stock en français), consiste à ne prendre en compte la correction que sur un exemple tiré aléatoirement à chaque itération. Modifiez le code en conséquence. Étudiez comme précédemment le comportement de l'algorithme. Quelles conclusions en tirez-vous ? Dans quel cas doit-on utiliser quelle variante ?
8. Question optionnelle : Étudier numériquement la vitesse de convergence dans le cas suivant : les X_i sont des points uniformément repartis sur les segments $\{0\} \times [0, M]$ (alors les y_i valent -1) ou bien sur le segment $\{\delta\} \times [0, M]$ (alors les y_i valent -1). De plus la proportion de de 1 est égal à $1/2$. On regardera l'impact de δ, M et n sur le temps de convergence du perceptron.
9. Proposez des variantes sur les conditions d'arrêt de l'algorithme.
10. Quel est le principal problème du perceptron ?
11. Trouver un fonction de perte telle que l'algorithme du perceptron soit équivalent à la version suivante (qui est la version initiale de l'algorithme) :

Pour en savoir plus sur le point de vue classique : hagan.okstate.edu/4_Perceptron.pdf
Enfin, pour un point de vue plus moderne sur la technique du gradient stochastique on peut consulter [3].

Perceptron : linéaire... vraiment ?

1. Quelle est la formule analytique d'une ellipse, d'une hyperbole et d'une parabole en 2D ?
2. Proposez une méthode pour réussir à classifier le jeu de données `clown`. Peut-on généraliser au delà ? On pourra utiliser la fonction `poly2` du fichier source associé au TP, qui plonge les données bi-dimensionnel dans l'espace des fonction polynomiale de degré 2 en les données. Comment l'utiliser pour apprendre un perceptron ?

Algorithme 2 : Perceptron "classique"

Data : les observations et leurs étiquettes $\mathcal{D}_n = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq n\}$; le pas de gradient : ϵ ; le nombre maximal d'itérations : n_{iter} ;

Result : $\mathbf{w}$

initialiser (aléatoirement) $\mathbf{w}$; initialiser $j = 0$

while $j \leq n_{\text{iter}}$ **do**

  $\mathbf{w}_{\text{old}} \leftarrow \mathbf{w}$

  $i = 0$

  while $j \leq n_{\text{iter}}$ *and* $\hat{f}_{\mathbf{w}_{\text{old}}}(\mathbf{x}_i) \cdot y_i \leq 0$ **do**

    $\mathbf{w} \leftarrow \mathbf{w}_{\text{old}} + \epsilon(1, \mathbf{x}_i) \cdot y_i$

    $j \leftarrow j + 1$

3. Sur le jeu de données `clown`, faites quelques expériences en transformant vos données et tracez les frontières de décision.

La méthode des k-plus proches voisins

L'algorithme des k-plus proches voisins (k-nn : pour *k-neighrest neighbors* en anglais) est un algorithme intuitif, aisément paramétrable et souvent performant pour traiter un problème de classification.

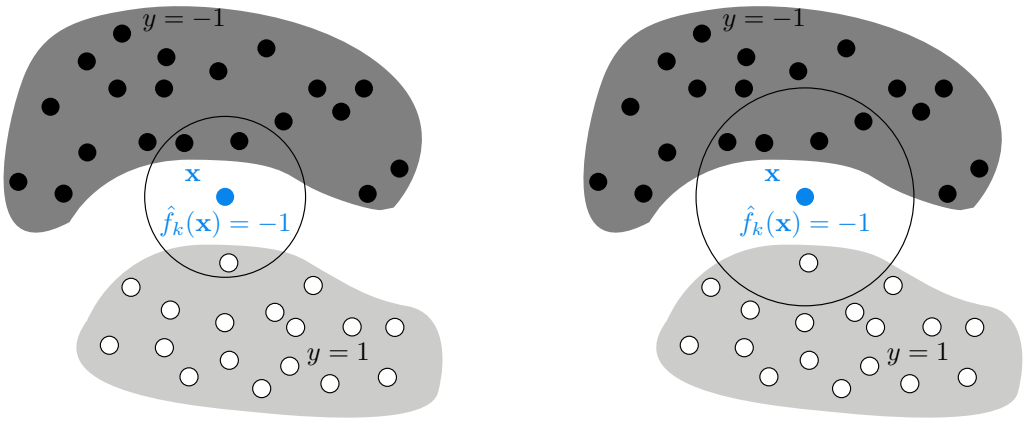
Le principe de l'algorithme est le suivant : pour chaque point $\mathbf{x}$ on commence par déterminer les k plus proches voisins $V_k(\mathbf{x}) = \{\mathbf{x}^1, \dots, \mathbf{x}^k\}$. La classe associée à $\mathbf{x}$ est la classe majoritaire dans $V_k(\mathbf{x})$.

De manière formelle, soit une distance $d : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$ et une fonction poids $\alpha : \mathbb{R}^+ \mapsto \mathbb{R}^+$ décroissante, la décision pour un point $\mathbf{x}$ est

$$\hat{f}_k(\mathbf{x}) \in \underset{y \in \{-1,1\}}{\arg\max} \left(\sum_{\mathbf{v} \in V_k(\mathbf{x}) \ : \ (\mathbf{v},y) \in \mathcal{D}_n} \alpha(d(\mathbf{x}, \mathbf{v})) \right).$$

Si α est constant cela revient au vote majoritaire parmi les voisins.

Nous utiliserons le paquet `sklearn.neighbors`. En suivant les mêmes étapes que précédemment, faites tourner sur un exemple cet algorithme de classification.



(a) $k = 3$ (b) $k = 5$

Figure 1 – Exemple des plus proches voisins pour deux valeurs du paramètres

4. Faites varier le nombre de voisins pris en compte. Quels sont les nombres remarquables ?
5. Expérimenter en comparant les frontières de décisions apprises : dans quelles cas la frontière est complexe ? simple ?
6. Un autre poids très utilisée est un poids gaussien, qui renvoie une valeur proportionelle à e^{-d}. À quel type de problème répond l'utilisation de poids variable en fonction de la distance ?
7. Quel est le taux d'erreur sur vos données d'apprentissage ? est-ce normal ? Que dire des prédictions sur de nouveaux exemples ? Quels points sont les plus sujet à erreurs ? Comment évaluer empiriquement mais de manière précise l'erreur sur les données artificielles ? Qu'en concluez vous pour des données réelles ?
8. Sur plusieurs jeux de données, sous plusieurs conditions de bruits, tracez les différentes courbes d'erreurs. Quel est la meilleure valeur de k ? Est-ce toujours la même ?
9. A votre avis, quels sont les avantages et les inconvénients des k-nn : optimalité ? temps de calcul ? expressivité ? passage à l'échelle ? interprétabilité ?
10. Appliquer la méthode k-nn (version multi-classe) aux données issues de la base zipcode de `scikitlearn` avec différents choix de $K \geq 1$. On pourra se référer à `http://scikit-learn.org/stable/_downloads/plot_digits_classification.py` pour le chargement et la manipulation de la base de donnée.

    Estimer la matrice de confusion $(\mathbb{P}\{Y = i, C_K(X) = j\})_{i,\,j}$ associée au classifieur C_K ainsi obtenu. Proposer une méthode pour choisir K et la mettre en oeuvre.

Arbres de régression - Algorithme CART

On pourra se référer à [2], chapitre 9.2 pour plus de détails sur les arbres. La source la plus détaillée sur le sujet étant le livre fondateur [1].

Avec `scikitlearn` on peut construire et élaguer des arbres de régression grâce au package `tree`.

```
from sklearn import tree
```

et créer un classifieur avec la fonction `tree.DecisionTreeClassifier`.

On considérera dans CART les mesures d'impureté suivantes :

- l'indice de Gini : $2\widehat{p}_k(R)(1-\widehat{p}_k(R))$
- l'entropie : $-\widehat{p}_k(R)\log(\widehat{p}_k(R)) - (1-\widehat{p}_k(R))\log(1-\widehat{p}_k(R))$.

1. Reprendre les exemples simulés précédemment, et jouer sur le paramètre de profondeur de l'arbre.
2. Tester les différents classifieurs obtenus sur de nouvelles données, et estimer leur risque.
3. Mettre en évidence le phénomène d'*overfitting* (ou sur-apprentissage en français) et l'équilibre biais-variance à trouver.
4. Effectuer le même genre de comparaison sur les données issues de la base zipcode.

Méthodes de choix de paramètres - Sélection de modèles

Il est rare de disposer en pratique d'un ensemble de test (on préfère inclure le plus grand nombre de données dans l'ensemble d'apprentissage). Pour sélectionner un modèle tout en considérant le plus grand nombre d'exemples possible pour l'apprentissage, on utilise généralement une procédure dite de sélection par validation croisée. Pour chaque paramétrisation du problème, une estimation de l'erreur empirique du classifieur appris est faîte selon la procédure suivante :

- l'ensemble d'apprentissage est partitionné en N ensembles
- pour les N combinaisons possibles, un classifieur est appris sur $N-1$ sous-ensembles et testé sur le dernier sous-ensemble
- l'erreur estimée est la moyenne de l'erreur des classifieurs appris.

Utiliser par exemple la fonction `sklearn.cross_validation.cross_val_score` et testez la en faisant varier le nombre de divisions envisagées. On pourra se servir de cette fonction avec pour choisir le nombre de voisins à considérer dans les méthodes des k plus proches voisins.

Références

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth Statistics/Probability Series. Wadsworth Advanced Books and Software, Belmont, CA, 1984. 8

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. 8

[3] S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2) :107–194, 2011. http://www.cs.huji.ac.il/~shais/papers/OLsurvey.pdf. 5