

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`

Pour avoir une version de Spyder la plus récente (ou d'autre éléments sous Anaconda) , taper dans un terminal :

```
cd /cal/softs/anaconda/anaconda-2.0.1/bin
```

puis taper

```
./spyder
```

- DÉCOUVERTE DE PYTHON -

Consulter les pages suivantes pour démarrer ou bien trouver quelques rappels

- *** http://perso.telecom-paristech.fr/~gramfort/lieesse_python/1-Intro-Python.html
- *** http://perso.telecom-paristech.fr/~gramfort/lieesse_python/2-Numpy.html
- *** http://perso.telecom-paristech.fr/~gramfort/lieesse_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 le 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 $Y : \forall i \in \{1, \dots, n\}, (\mathbf{x}_i, y_i) \stackrel{i.i.d}{\sim} (X, Y)$.
- 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. Stockez quelques jeux de données afin de les utiliser dans la suite : pour chacun, il faudra sauver 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.

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 $\mathbf{w} = (w_0, w_1, \dots, w_p) \in \mathbb{R}^{p+1}$ 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 $\text{sign}(\hat{f}_{\mathbf{w}}(\mathbf{x}))$, 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 supposons 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 $\text{sign}(\hat{f}_{\mathbf{w}}(\mathbf{x}))$.

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 $C_{\mathbf{w}} = \mathbb{E} [\ell(\hat{f}_{\mathbf{w}}(\mathbf{x}), y)]$ 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 `frontiere` par exemple) le coût sur une base d'exemple en fonction de \mathbf{w} . Où se situe le vecteur

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

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 *Stochastic 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) :

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

```

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 [2].

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

Analyse Discriminante Linéaire

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

Aspect théorique (en première lecture passer les questions 2-6)

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 $\Sigma = \mathbb{E}[(X - \mathbb{E}(X))(X - \mathbb{E}(X))^\top]$.

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 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ 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 $\hat{\pi}_+$, $\hat{\mu}_+$, $\hat{\mu}_-$ et $\hat{\Sigma}$ des paramètres.
4. Justifier le choix du classifieur suivant :

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

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

On suppose $K = 2$, et l'on code les classes par 0 et 1. On souhaite modéliser les probabilités conditionnelles des classes, ou plutôt leur log-ratio, par des quantités linéaires (affines) :

$$\log \left(\frac{\mathbb{P}(Y = 1|X = \mathbf{x})}{\mathbb{P}(Y = 0|X = \mathbf{x})} \right) = \alpha + \langle \beta, \mathbf{x} \rangle$$

où $\alpha \in \mathbb{R}$ et $\beta \in \mathbb{R}^p$

Sous une telle hypothèse la séparatrice est **linéaire**, la règle étant simplement :

$$\alpha + \langle \beta, \mathbf{x} \rangle > 0$$

signifiant que l'on préfère la classe 0 à la classe 1 pour le point \mathbf{x} .

On peut alors estimer les probabilités conditionnelles facilement :

$$\begin{aligned} \mathbb{P}(Y = 1|X = \mathbf{x}) &= \frac{\exp(\alpha + \langle \beta, \mathbf{x} \rangle)}{1 + \exp(\alpha + \langle \beta, \mathbf{x} \rangle)} \\ \mathbb{P}(Y = 0|X = \mathbf{x}) &= \frac{1}{1 + \exp(\alpha + \langle \beta, \mathbf{x} \rangle)} \end{aligned}$$

On cherche alors à maximiser la vraisemblance

$$\begin{aligned} \ell(\alpha, \beta) &= \sum_{i=1}^n \log(\mathbb{P}(Y = Y_i|X = X_i, \alpha, \beta)) \\ &= \sum_{i=1}^n \sum_{k \in \{0,1\}} \mathbb{1}_{\{Y_i=k\}} \log(\mathbb{P}(Y = k|X = X_i, \alpha, \beta)) \end{aligned}$$

on cherche alors une solution

$$(\hat{\alpha}, \hat{\beta}) \in \arg \max_{\alpha, \beta} \ell(\alpha, \beta)$$

$$\text{où } \ell(\alpha, \beta) = \sum_{i=1}^n \left(Y_i(\alpha + \langle \beta, X_i \rangle) - \log[1 + \exp(\alpha + \langle \beta, X_i \rangle)] \right)$$

Numeriquement ce problème d'optimisation peut être résolu par différentes méthodes (le défaut en scikitlearn étant : [1] <http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf>, ou plus simplement la méthode de Newton)

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
```

Nous allons voir en détail le processus pour utiliser 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? `intercept_`?
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>.

Références

- [1] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear : A library for large linear classification. *J. Mach. Learn. Res.*, 9 :1871–1874, 2008. 6
- [2] 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>. 4