Apprentissage statistique
TP4 : Marges et noyaux

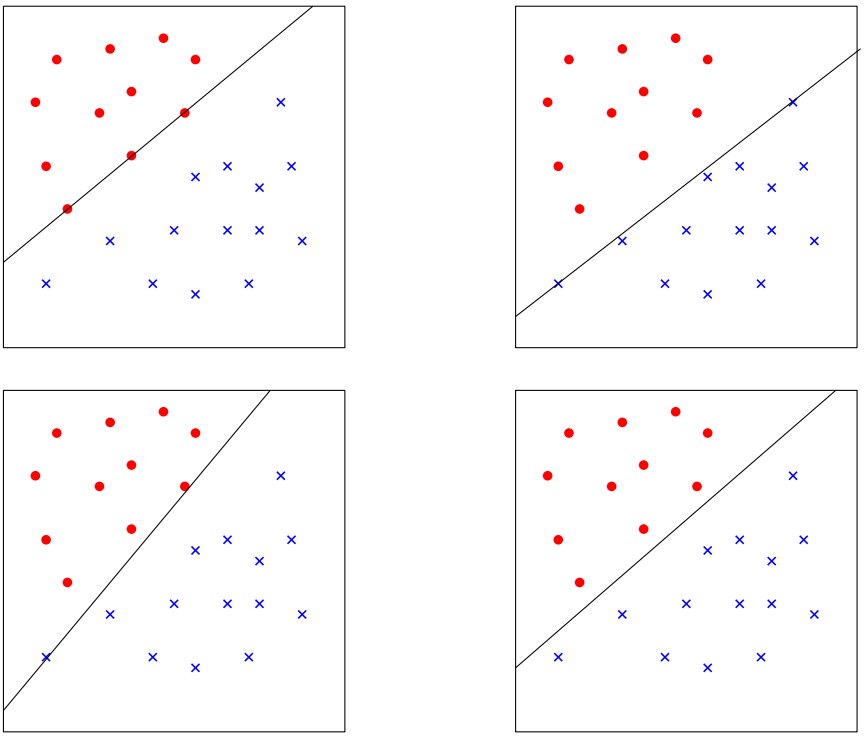
Olivier Schwander <olivier.schwander@lip6.fr>

2018-2019

Exercice 1 *Introduction*

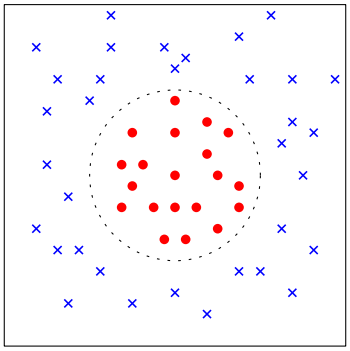
Question 1

Commenter les hyperplans séparateurs dans les figures ci-dessous. Lequel est le plus intéressant dans un contexte de classification ?



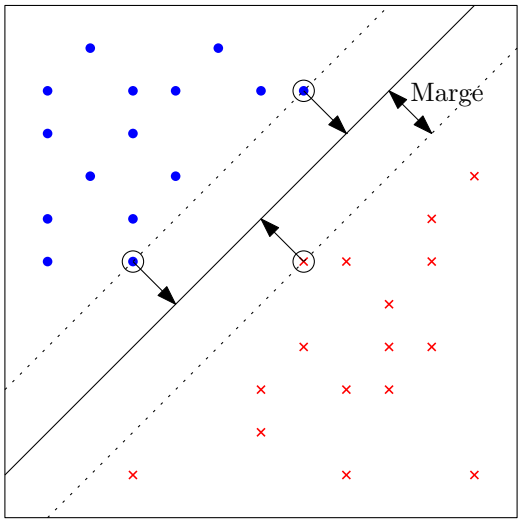
Question 2

Les deux classes dans la figure ci-dessus ne sont pas linéairement séparables. Proposer un changement de variable pour les rendre linéairement séparables.



Exercice 2 *Machines à vecteurs supports*

Les machines à vecteurs supports (*Support Vector Machines*, SVM) cherchent l'hyperplan séparateur qui maximise la marge entre l'hyperplan et chacune des deux classes.



Soit un ensemble d'apprentissage $\{x_1, \dots, x_N\} \in \mathbb{R}^d$ et les étiquettes associées $\{l_1, \dots, l_N\} \in \{-1, +1\}$.

L'hyperplan optimal, d'équation $w^T x + w_0 = 0$ est la solution du problème d'optimisation suivant :

$$\max_{w, w_0} \min_{k} \{\|x - x_k\| \text{ tq } w^T x + w_0 = 0\}$$

Question 1

Calculer la distance entre un point et l'hyperplan séparateur. Réécrire le problème d'optimisation.

Question 2

On choisit de renormaliser le vecteur normal de façon à obtenir pour tous les points x_{marge} situés sur la marge :

$$l_{x_{\text{marge}}} \left(w^T x_{\text{marge}} + w_0\right) = 1$$

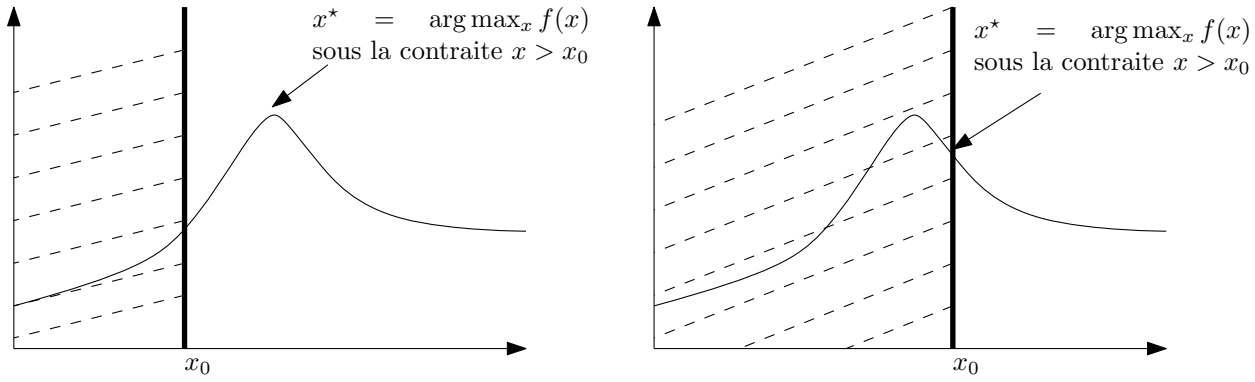
On a donc pour toutes les observations :

$$l_k \left(w^T x_k + w_0 \right) \geq 1$$

Exprimer le problème sous la forme d'une minimisation sous contrainte par rapport à w, w_0.

Question 3

Quelle est l'influence de la contrainte d'inégalité dans les deux problèmes d'optimisation décrits dans les figures suivantes ? Dans quelle mesure la solution dépend-elle de la contrainte ?



Question 4

A-t-on besoin de tous les points du jeu de données lors du calcul de l'hyperplan qui maximise la marge ?

Question 5

En appliquant la méthodes des multiplicateurs de Lagrange et les conditions KKT, on obtient le problème dual suivant :

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j l_i l_j x_i x_j \text{ sous les contraintes } \alpha_k \geq 0 \text{ et } \sum_k \alpha_k l_k = 0$$

A-t-on besoin de connaître directement les vecteurs x_k représentant les observations pour résoudre ce problème ?

Question 6

Quelle partie des conditions KKT permet de retrouver la propriété de la question 4 ?

Exercice 3 *Astuce du noyau*

L'astuce du noyau (*kernel trick*) repose en premier lieu sur le fait que la solution du problème d'optimisation précédent ne dépend que du produit scalaire des vecteurs et pas des vecteurs eux-mêmes. On va utiliser cette remarque pour appliquer SVM à des cas non-linéairement séparables, en travaillant non pas sur les observations directement mais sur une version plongée dans un espace de dimension supérieure, dans lequel on espère que les classes sont linéairement séparables.

Le problème devient alors :

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j l_i l_j \phi(x_i)\phi(x_j) \text{ sous les contraintes } \alpha_k \geq 0 \text{ et } \sum_k \alpha_k l_k = 0$$

où ϕ est la fonction pour aller de l'espace d'origine vers l'espace de plus grande dimension.

Question 1

Comparer le coût de calcul du produit scalaire dans l'espace d'origine (de dimension d) et dans l'espace du plongement (de dimension $D \gg d$).

Question 2

En second lieu, l'application du théorème de Mercer permet de calculer des produits scalaires dans un espace de grande dimension, en utilisant une fonction noyau, sans calculer explicitement les vecteurs plongés, ni la fonction ϕ, ou même la dimension précise de l'espace du plongement.

Réécrire la minimisation en exploitant une fonction noyau K pour calculer le produit scalaire.

Exercice 4 *Application*

Les expériences se feront d'abord sur des données jouet (écrites à la main, ou générées à partir de deux lois gaussiennes), puis sur la base d'images USPS décrite lors du TP2, en réutilisant le module `usps.py`.

La bibliothèque `sklearn` contient une implantation de l'algorithme SVM (reposant sur la bibliothèque de référence `libsvm`). Son utilisation est très proche de celle du perceptron :

```
from sklearn import svm

data   = [[0, 0], [1, 1]]
labels = [-1, 1]

classifier = svm.SVC()
classifier.fit(data, labels)
predicted = classifier.predict(x)
```

Question 1

Sur les données jouet, afficher les vecteurs supports ainsi que l'hyperplan.

On peut récupérer la liste des vecteurs supports avec :

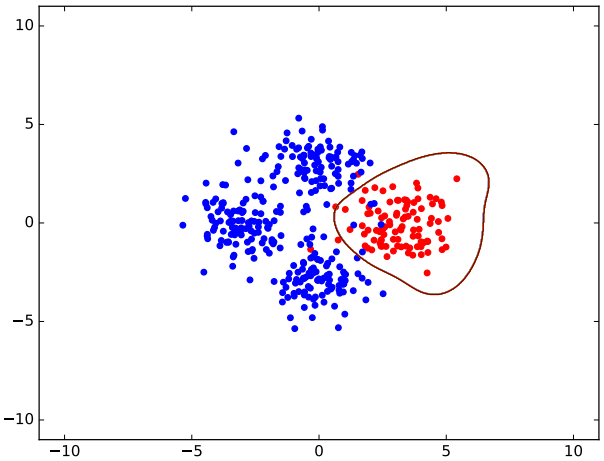
```
classifier.support_vectors_
```

Question 2

Les noyaux disponibles dans `sklearn` sont les suivants :

- linéaire : $\langle x, x \rangle$ (argument `kernel='linear'` ;
- polynomial : $\gamma\langle x, x' \rangle + r)^d$ (`kernel='poly'`, `degree=`d,`coef0=`r) ;
- gaussien (RBF, pour *Radial Basis Function*) : exp(- |x-x'|^2) (`kernel=rbf'`, `gamma=`γ).

Sur les données jouet, tracer les régions associées aux différentes classes, avec plusieurs noyaux.



Question 3

Comparer les résultats de classification avec plusieurs noyaux.

On pourra utiliser des fonctions du modules `sklearn.metrics` :

- `classification_report(expected, predicted)`,
- `metrics.confusion_matrix(expected, predicted)`.