TP2

March 10, 2017

Loïc Herbelot

1 TP 2 : SVM

1.1 Question 1) Classification linéaire des données iris

In [48]:
```
# -*- coding: utf-8 -*-
"""
@author: Loïc Herbelot
"""

from sklearn import datasets
from sklearn.svm import SVC
import numpy as np
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

"""This data sets consists of 3 different types of irises
(Setosa, Versicolour, and Virginica) petal and sepal length,
stored in a 150x4 numpy.ndarray
The rows being the samples and the columns being:
Sepal Length, Sepal Width, Petal Length and Petal Width."""


X = iris.data
y = iris.target

#We only want classes 1 & 2, and consider only the first 2 features.
X = X[y != 0, :2]
y = y[y != 0]

#Shuffling the data:
permutation = np.random.permutation(len(X))
X = X[permutation]
y = y[permutation]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5)
```

In [49]:
```
# fit the model with linear kernel
clf_lin = SVC(kernel='linear')
clf_lin.fit(X_train, y_train)

# predict labels for the test data base
y_pred = clf_lin.predict(X_test)

# check your score
score = clf_lin.score(X_test, y_test)
print('Score with linear kernel: %s' % score)
```

```
Score with linear kernel: 0.68
```

1.2 Question 2) Classification polynomiale

In [50]:
```
clf_poly = SVC(kernel='poly')
clf_poly.fit(X_train, y_train)

# predict labels for the test data base
y_pred = clf_poly.predict(X_test)

# check your score
score = clf_poly.score(X_test, y_test)
print('Score with polynomial kernel: %s' % score)
```

```
Score with polynomial kernel: 0.66
```

1.3 Question 3) Réécriture du problème primal :

Dans le problème primal, on a les contraintes :
$\xi_i \geq 0$ et $\xi_i \geq 1 - y_i(w \cdot \Phi(x_i) + w_0)$
Ainsi $\xi_i \geq max(0, 1 - y_i(w \cdot \Phi(x_i) + w_0))$
Donc $\xi_i = [1 - y_i(w \cdot \Phi(x_i) + w_0)]_+$
D'où la réécriture du problème primal.

1.4 Question 4) Explication du SVM

Si après avoir trouvé le vecteur w, on a une erreur de prédiction sur le point x_i, alors la marge qui vaut $marge_i = y_i(w \cdot \Phi(x_i) + w_0)$ est négative, ainsi $\xi_i = 1 - marge_i \geq 1$.

Sinon, si la prédiction est correcte, la marge est positive et $\xi_i = 0$.

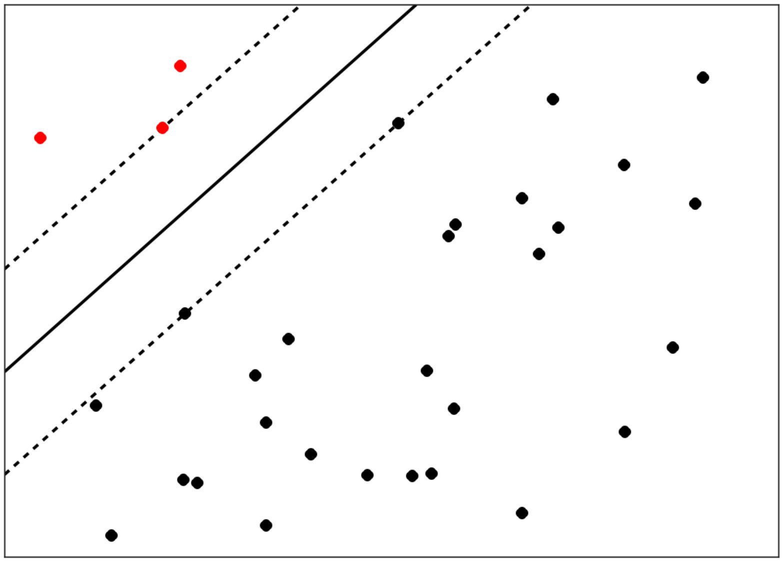
Ainsi il nous faudrait une fonction qui représente l'erreur de classification, qui vaut au moins 1 quand la marge est négative (cas d'erreur), et 0 quand la marge est positive (prédiction correcte).

Pour que les calculs soient plus pratiques, cette fonction est convexe.

La fonction charnière (*hinge*) correspond aux caractéristiques voulues, et le SVM tente de minimiser l'image de cette fonction.

1.5 SVM GUI avec des classes déséquilibrées :

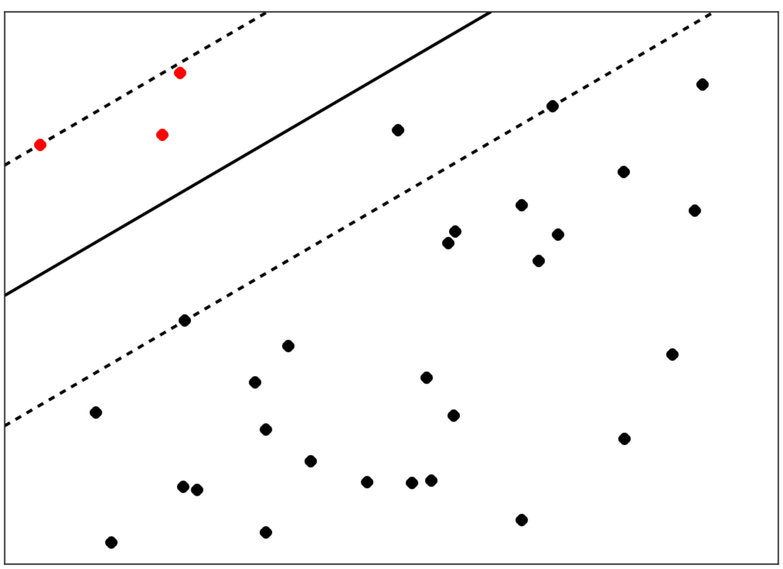
1.5.1 Avec $C = 1$



Linear: $u^T v$ RBF: $\exp(-\gamma|u-v|^2)$ Poly: $(\gamma\ u^T\ v + r)^d$

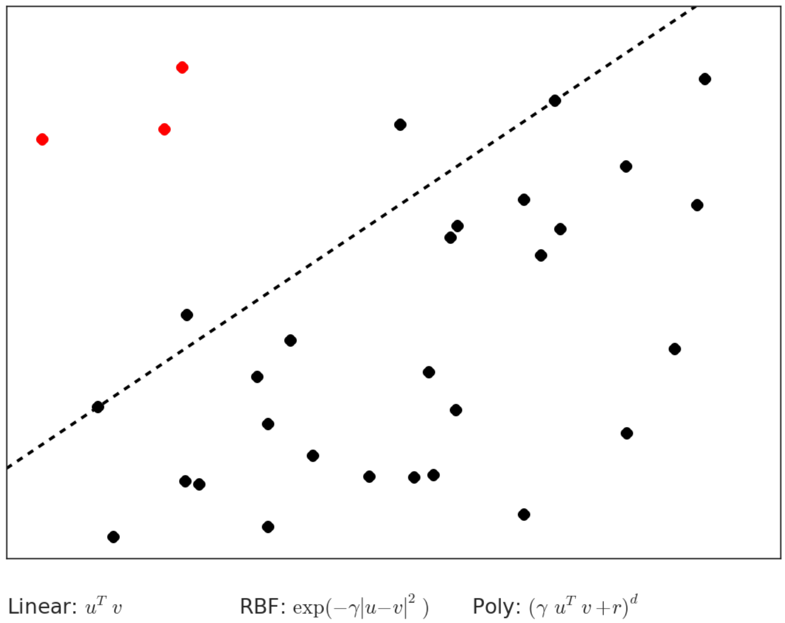
title

1.5.2 Avec $C = 0.001$



Linear: $u^T v$ RBF: $\exp(-\gamma|u-v|^2)$ Poly: $(\gamma\ u^T\ v + r)^d$

Avec C =



0.00001

On voit que lorsque le paramètre de régularisation C devient très fabible, la classe la plus représentée ń écrase ż l'autre classe.

```
In [ ]:
```