TELECOM ParisTech **SD210**
2016/2017 Balamurugan Palaniappan, Alexandre Garcia, Moussab Djerrab, Pietro Gori

---

TP n° 2 : Support Vector Machine (SVM)

Liens utiles pour ce TP :

⋆⋆⋆ http://scikit-learn.org/stable/modules/svm.html

⋆⋆ http://en.wikipedia.org/wiki/Support_vector_machine

⋆⋆ http://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support

- Introduction et fondements mathématiques -

Les SVM ont été introduites par Vapnik [3], et sont abordées au chapitre 12 du livre [1]. Des détails plus mathématiques peuvent être trouvés dans le chapitre 7 du livre [2]. La popularité des méthodes SVM, pour la classification binaire en particulier, provient du fait qu'elles reposent sur l'application d'algorithmes de recherche de règles de décision linéaires : on parle d'hyperplans (affine) séparateurs. Toutefois, cette recherche s'effectue dans un espace de caractères (*feature space*, en anglais) de très grande dimension qui est l'image de l'espace d'entrée original par une transformation Φ non linéaire.

Le but de ce TP est de mettre en pratique ce type de techniques de classification sur données réelles et simulées au moyen du package `scikit-learn` (lequel met en œuvre la librairie en `C` LIBSVM) et d'apprendre à contrôler les paramètres garantissant leur flexibilité (hyper-paramètres, noyau).

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 une observation (ou un exemple),
- $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$ 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 $\mathbf{x}$ (*i.e.,* qui n'est pas présent dans l'ensemble d'apprentissage) prédit son étiquette : $\hat{f}(\mathbf{x})$. La règle que l'on considère ici est dite linéaire, dans le sens où l'on sépare l'espace par un hyperplan (affine) : selon la position par rapport à celui-ci, on prédit alors $+1$ ou -1.

SVM et noyaux pour la classification binaire

Les techniques SVM (non linéaires) font appel à une fonction implicite Φ transformant l'espace d'entrée $\mathcal{X} \subset \mathbb{R}^p$ en un espace hilbertien $(\mathcal{H}, \langle \cdot, \cdot \rangle)$ de plus grande dimension. L'apprentissage s'effectue alors à partir du modèle $(\Phi(X), Y)$ dans l'espace $\mathcal{H}$, de dimension plus grande certes, mais dans lequel on espère que les données soient "davantage linéairement séparables". Du point de vue pratique, il convient de noter que le calcul des projections $\Phi(X)$ n'est pas utilisé dans la méthode, seuls les produits scalaires $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$, $(\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2$, sont requis. Or, ceux-ci sont donnés par un noyau K, via la relation *kernel trick* (en anglais : astuce du noyau) :

? ? ?

$$K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle.$$

La méthode requiert donc de sélectionner un noyau (ainsi que d'autre paramètres). Parmi les choix possibles, on compte en particulier les noyaux suivants

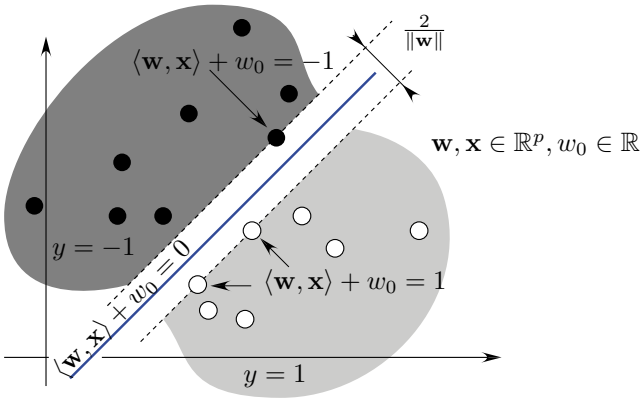

FIGURE 1 – Marge et hyperplan séparateur dans le cadre de classes séparables (cas d'un noyau linéaire)

- Noyau linéaire : $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ (correspondant aux SVM linéaires)
- Noyau Gaussien radial (Gaussian RBF) $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$ Radial basis function kernel
- Noyaux polynomiaux $K(\mathbf{x}, \mathbf{x}') = (\alpha + \beta \langle \mathbf{x},\ \mathbf{x}' \rangle)^\delta$ pour un $\delta > 0$
- Noyau radial de Laplace (Laplace RBF) $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|\right)$ pour un $\gamma > 0$
- Noyau tangente hyperbolique (sigmoïde) $K(\mathbf{x}, \mathbf{x}') = \tanh\left(\alpha + \beta \langle \mathbf{x},\ \mathbf{x}' \rangle\right)$ pour un couple $(\alpha, \beta) \in \mathbb{R}^2$

Un classifieur SVM est de la forme

$$\hat{f}_{\mathbf{w}, w_0}(\mathbf{x}) = \operatorname{sign}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + w_0\right), \tag{1}$$

où $\mathbf{w} \in \mathcal{H}$ et $w_0 \in \mathbb{R}$ sont des paramètres ajustés lors de la phase d'apprentissage à partir d'un échantillon d'exemples i.i.d., $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$. À noter : dans le cas où la fonction Φ est l'identité sur $\mathbb{R}^p$, on trouve simplement que la règle de décision est

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

La frontière associée à la règle de décision (1) est l'ensemble $\{\mathbf{x} : \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + w_0 = 0\}$. Elle correspond à un hyperplan dans l'espace $\mathcal{H}$, mais est beaucoup plus complexe dans $\mathcal{X}$ (selon la forme du noyau choisi).

Dans $\mathcal{H}$, l'hyperplan est obtenu en maximisant la marge séparant les deux classes, ce qui revient à résoudre un problème d'optimisation sous contraintes linéaires :

$$\begin{cases} (\mathbf{w}^*, w_0^*, \xi^* \in \mathbb{R}^n) \in \underset{\mathbf{w} \in \mathcal{H}, w_0 \in \mathbb{R}, \xi \in \mathbb{R}^n}{\arg\min} \left(\dfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i\right) & \\ \text{s.c} \quad \xi_i \geq 0, & \forall i \in \{1, \cdots, n\}, \\ \qquad y_i \left(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle + w_0\right) \geq 1 - \xi_i, & \forall i \in \{1, \cdots, n\}. \end{cases} \tag{2}$$

On peut montrer que la solution $\mathbf{w}$ peut s'exprimer de la façon suivante :

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \Phi(\mathbf{x}_i).$$

Les indices i pour lesquels $\alpha_i^* \neq 0$ sont ceux pour lesquels l'égalité est réalisée dans la contrainte, les points $\mathbf{x}_i$ correspondants sont appelés *support vectors* (vecteurs supports en français) de la décision. Les

coefficients α_i^* désignent les solutions du problème dual qui est un programme quadratique (QP) sous contraintes affines :

$$\begin{cases} \alpha^* \in \underset{\alpha \in \mathbb{R}^n}{\arg\max} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{1 \leq i,\, j \leq n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) \\ \text{s.c} \quad 0 \leq \alpha_i \leq C, \quad \forall i \in \{1, \cdots, n\}, \\ \qquad \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases} \tag{3}$$

Le paramètre C contrôle la complexité du classifieur dans la mesure où il détermine le coût d'une mauvaise classification : plus C est grand, plus la règle obtenue est complexe (le nombre de points pour lesquels on veut minimiser l'erreur de classification croît). Cette approche est appelée C-classification et s'utilise avec l'objet `sklearn.svm.SVC` dans le module `scikit-learn`.

Une autre façon de contrôler la complexité (*i.e.,* le nombre de vecteurs supports), appelée ν-classification, revient à considérer, à la place du problème dual décrit ci-dessus, le problème suivant :

$$\begin{cases} \alpha^* \in \underset{\alpha \in \mathbb{R}^n}{\arg\min} \left(\frac{1}{2} \sum_{1 \leq i,\, j \leq n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right) \\ \text{s.c} \quad 0 \leq \alpha_i \leq 1, \quad \forall i \in \{1, \cdots, n\}, \\ \qquad \sum_{i=1}^{n} \alpha_i y_i = 0, \\ \qquad \sum_{i=1}^{n} \alpha_i \geq \nu. \end{cases} \tag{4}$$

où $\nu \in [0, 1]$ est un paramètre approchant le pourcentage de vecteurs supports parmi les données d'apprentissage. Cette approche s'utilise avec l'objet `sklearn.svm.NuSVC` dans le module `scikit-learn`.

Extensions au 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 $K(K-1)/2$) 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 $Y \neq k$. A partir des estimations des probabilités a posteriori, on affecte le label estimé le plus probable.

- Mise en oeuvre -

On utilisera l'objet `sklearn.svm.SVC` :

```
from sklearn.svm import SVC
```

Le fichier `svm_script.py` vous fournit un exemple complet de classification utilisant la fonction `SVC`. Utilisez cet exemple pour vous familiariser avec la syntaxe puis reproduisez une expérience similaire avec les données `iris` comme suggéré ci-dessous.

1) En vous basant sur la documentation à l'adresse suivante :

   http://scikit-learn.org/stable/modules/svm.html

   écrivez un code qui va classifier la classe 1 contre la classe 2 du dataset `iris` en utilisant les deux premières variables et un noyau linéaire. En laissant la moitié des données de côté, évaluez la performance en généralisation du modèle. Le dataset `iris` s'obtient avec les lignes suivantes :

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
X = X[y != 0, :2]
y = y[y != 0]
```

2) Comparez le résultat avec un SVM basé sur noyau polynomial.
3) Montrez que le problème primal résolu par le SVM peut se réécrire de la façon suivante :

$$\underset{\mathbf{w}\in\mathcal{H},w_0\in\mathbb{R}}{\arg\min}\left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\left[1 - y_i\left(\langle\mathbf{w},\ \Phi(\mathbf{x}_i)\rangle + w_0\right)\right]_+\right)$$

4) Expliquez la phrase : "un SVM minimise l'erreur de classification à l'aide d'un majorant convexe de la fonction qui vaut 1 quand la marge est négative et 0 sinon". La fonction $x \to [1-x]_+ = \max(0, 1-x)$ est appelée *Hinge* (charnière en français).

SVM GUI

— Lancez le script `svm_gui.py` disponible à l'adresse : http://scikit-learn.org/stable/auto_examples/applications/svm_gui.html Cette application permet en temps réel d'évaluer l'impact du choix du noyau et du paramètre de régularisation C.

— Générez un jeu de données très déséquilibré avec beaucoup plus de points dans une classe que dans l'autre (au moins 90% vs 10%).

— A l'aide d'un noyau linéaire et en diminuant le paramètre C qu'observez vous ?
Ce phénomène peut être corrigé en pratique en pondérant d'avantage les erreurs sur la classe la moins présente (paramètre `class_weight` de SVC) ou par une technique de re-calibration (utilisée avec `SVC(..., probability=True)`).

Classification de visages

L'exemple suivant est un problème de classification de visages. La base de données à utiliser est disponible à l'adresse suivante : http://perso.telecom-paristech.fr/~gramfort/lfw.zip .

En modifiant l'exemple de code donné dans la dernière partie du fichier `svm_script.py` :

5) montrez l'influence du paramètre de régularisation. On pourra par exemple afficher l'erreur de prédiction en fonction de C sur une échelle logarithmique entre `1e5` et `1e-5`.
6) en ajoutant des variables de nuisances, augmentant ainsi le nombre de variables à nombre de points d'apprentissage fixé, montrez que la performance chute.
7) Expliquez pourquoi les features sont centrées et réduites `l.148-150`.
8) quel est l'effet du choix d'un noyau non-linéaire RBF sur la prédiction ? Vous pourrez améliorer la prédiction à l'aide d'une réduction de dimension basée sur l'objet `sklearn.decomposition.RandomizedPCA`.

Pour aller plus loin : Calcul du saut de dualité

L'exemple suivant :

http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#example-svm-plot-separating-hyperplane-py

explique comment accéder aux paramètres estimés par le modèle (vecteur de coefficients $\mathbf{w}$ dans l'attribut `coef_`, w_0 attribut `intercept`, liste des vecteurs de supports, coefficients du problème dual).

9) En vous basant sur cet exemple écrivez un code qui va calculer la valeur des fonctionnelles primales et duales et vérifiez que les valeurs sont proches. Vous afficherez sur un même graphe les valeurs des fonctionnelles.

10) Comment varie la différence entre les deux valeurs quand on fait varier la tolérance sur l'optimisation (paramètre `tol` de `SVC`) ?

```
"""
===================================================
Faces recognition example using SVMs and custom kernels
===================================================

The dataset used in this example is a preprocessed excerpt
of the "Labeled Faces in the Wild", aka LFW_:

  http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz (233MB)

  _LFW: http://vis-www.cs.umass.edu/lfw/

"""

import numpy as np
from time import time
import pylab as pl

from sklearn.cross_validation import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.svm import SVC

###############################################################################
# Download the data (if not already on disk); load it as numpy arrays
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4,
                              color=True, funneled=False, slice_=None,
                              download_if_missing=True)
# data_home='.'

# introspect the images arrays to find the shapes (for plotting)
images = lfw_people.images / 255.
n_samples, h, w, n_colors = images.shape

# the label to predict is the id of the person
target_names = lfw_people.target_names.tolist()

###############################################################################
# Pick a pair to classify such as
names = ['Tony Blair', 'Colin Powell']
# names = ['Donald Rumsfeld', 'Colin Powell']

idx0 = (lfw_people.target == target_names.index(names[0]))
idx1 = (lfw_people.target == target_names.index(names[1]))
images = np.r_[images[idx0], images[idx1]]
n_samples = images.shape[0]
y = np.r_[np.zeros(np.sum(idx0)), np.ones(np.sum(idx1))].astype(np.int)

###############################################################################
# Extract features

# features using only illuminations
X = (np.mean(images, axis=3)).reshape(n_samples, -1)

# # or compute features using colors (3 times more features)
```

```
# X = images.copy().reshape(n_samples, -1)

# Scale features
X -= np.mean(X, axis=0)
X /= np.std(X, axis=0)

###############################################################################
# Split data into a half training and half test set
# X_train, X_test, y_train, y_test, images_train, images_test = \
#    train_test_split(X, y, images, test_size=0.5, random_state=0)
# X_train, X_test, y_train, y_test = \
#    train_test_split(X, y, test_size=0.5, random_state=0)

indices = np.random.permutation(X.shape[0])
train_idx, test_idx = indices[:X.shape[0] / 2], indices[X.shape[0] / 2:]
X_train, X_test = X[train_idx, :], X[test_idx, :]
y_train, y_test = y[train_idx], y[test_idx]
images_train, images_test = images[
    train_idx, :, :, :], images[test_idx, :, :, :]

###############################################################################
# Quantitative evaluation of the model quality on the test set
print "Fitting the classifier to the training set"
t0 = time()
clf = SVC(kernel='linear', C=1.0)
clf = clf.fit(X_train, y_train)

print "Predicting the people names on the testing set"
t0 = time()
y_pred = clf.predict(X_test)

print "done in %0.3fs" % (time() - t0)
print "Chance level : %s" % max(np.mean(y), 1. - np.mean(y))
print "Accuracy : %s" % clf.score(X_test, y_test)

###############################################################################
# Look at the coefficients
pl.figure()
pl.imshow(np.reshape(clf.coef_, (h, w)))

###############################################################################
# Qualitative evaluation of the predictions using matplotlib


def plot_gallery(images, titles, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    pl.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    pl.subplots_adjust(bottom=0, left=.01, right=.99, top=.90,
                       hspace=.35)
    for i in range(n_row * n_col):
        pl.subplot(n_row, n_col, i + 1)
        pl.imshow(images[i])
        pl.title(titles[i], size=12)
        pl.xticks(())
        pl.yticks(())
```

```
def title(y_pred, y_test, names):
    pred_name = names[int(y_pred)].rsplit(' ', 1)[-1]
    true_name = names[int(y_test)].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:   %s' % (pred_name, true_name)

prediction_titles = [title(y_pred[i], y_test[i], names)
                     for i in range(y_pred.shape[0])]
plot_gallery(images_test, prediction_titles)
pl.show()
```

Références

[1] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. http://www-stat.stanford.edu/~tibs/ElemStatLearn/. 1

[2] B. Schölkopf and A. J. Smola. *Learning with kernels : Support vector machines, regularization, optimization, and beyond*. MIT press, 2002. 1

[3] Vladimir N Vapnik. *Statistical learning theory*. Wiley, 1998. 1