
TP N° 1 : Support Vector Machine (SVM)

Useful links for this lab :

★★★ 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 and Mathematical Foundations -

The SVMs were introduced by Vapnik [3], and are discussed in Chapter 12 of the book [1]. More mathematical details can be be found in Chapter 7 of the book [2]. The popularity of SVM methods, for binary classification in particular, comes from the fact that they rely on the application of algorithms of linear decision rules : we speak of hyperplanes (affine) separators. However, this search takes place in a higher-dimensional *feature space*, that is the image of the original input space transformed by a non-linear transformation Φ.

The aim of this TP is to put into practice this type of classification techniques on actual and simulated data in the `scikit-learn` package (which implements the library in `C` LIBSVM) and to learn to control the parameters guaranteeing their flexibility (hyper-parameters, kernel).

Definitions and Notations

It is recalled that within the framework of the supervised binary classification we use the notations :

— $\mathcal{Y}$ The set of labels, commonly $\mathcal{Y} = \{-1, 1\}$ in the case of binary classification,

— $\mathbf{x} = (x_1, \dots, x_p) \in \mathcal{X} \subset \mathbb{R}^p$ is an observation (or an example),

— $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$ a learning set containing n examples and their labels,

— There is a probabilistic model that governs the generation of our observations according to random variables X and Y : $\forall i \in \{1, \dots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.

— We try to build from the learning set $\mathcal{D}_n$ a function $\hat{f} : \mathcal{X} \mapsto \{-1, 1\}$ which for an unknown point $\mathbf{x}$ (*i.e.*, which is not present in the learning set) predicts its label : $\hat{f}(\mathbf{x})$. The rule considered here is called linear, in the sense that the space is separated by a hyperplane (affine) : according to the position with respect thereto, then predicting $+1$ or -1.

SVM and kernels for binary classification

SVM (nonlinear) techniques use an implicit function Φ transforming the input space $\mathcal{X} \subset \mathbb{R}^p$ in a Hilbert space $(\mathcal{H}, \langle \cdot, \cdot \rangle)$ of higher dimension. Learning is then carried out using the model $(\Phi(X), Y)$ in the space $\mathcal{H}$, of larger size of course, but in which it is hoped that the data is "more linearly separable". From a practical point of view, It should be noted that the calculation of projections $\Phi(X)$ is not used in the method, only scalar products $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$, $(\mathbf{x}, \mathbf{x}') \in \mathcal{X}^2$, are required. Now these are given by a kernel K, via the relation *kernel trick* :

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

The method therefore requires selecting a kernel (as well as other parameters). Among the possible choices, there are in particular the following kernels

- Linear Kernel : $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ (corresponding to linear SVMs)

- Gaussian Kernel (Gaussian RBF) $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$

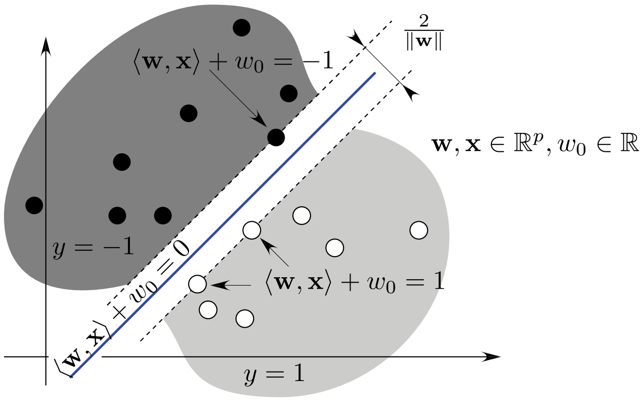

FIGURE 1 – Margin and separator hyperplane in separable classes (case of a linear kernel)

- Polynomial Kernel $K(\mathbf{x}, \mathbf{x}') = (\alpha + \beta\langle \mathbf{x},\ \mathbf{x}' \rangle)^\delta$ for a $\delta > 0$
- Laplacian Kernel (Laplace RBF) $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|)$ for a $\gamma > 0$
- Hyperbolic tangent kernel (Sigmoid) $K(\mathbf{x}, \mathbf{x}') = \tanh(\alpha + \beta\langle \mathbf{x},\ \mathbf{x}' \rangle)$ for a pair $(\alpha, \beta) \in \mathbb{R}^2$

A SVM classifier is of the form

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

or $\mathbf{w} \in \mathcal{H}$ and $w_0 \in \mathbb{R}$ are parameters adjusted during phase learning from a sample of i.i.d. examples, $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots n\}$. Note : in the case where the funcion Φ is the identity on $\mathbb{R}^p$, we simply find that the decision rule is

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

The boundary associated with the decision rule (1) is the set $\{\mathbf{x} : \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + w_0 = 0\}$. It corresponds to a hyperplane in space $\mathcal{H}$, but is much more complex in $\mathcal{X}$ (depending on the shape of the selected kernel).

In $\mathcal{H}$, the hyperplane is obtained by maximizing the margin separating the two classes. This requires to solve an optimization problem under linear constraints :

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

It can be shown that the solution $\mathbf{w}$ can be expressed in the following way :

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

The indices i for which $\alpha_i^* \neq 0$ are those for whom equality is carried out in the constraint, the corresponding points $\mathbf{x}_i$ are called *support vectors* (vecteurs supports en français) of the decision. The coefficients α_i^* denote the solutions of the dual problem which is a quadratic program (QP) under affine constraints :

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

The parameter C controls the complexity of the classifier insofar as it determines the cost of a misclassification : the higher the C value, the more the rule obtained is complex (the number of points for which one wants to minimize the classification error crop). This approach is called C-classification and is used with the object `sklearn.svm.SVC` in the module `scikit-learn`.

Another way to control complexity (*i.e.,* the number of support vectors), called ν-classification, is to consider, in place of the dual problem described above, the following problem :

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

where $\nu \in [0, 1]$ is a parameter approximating the percentage of support vectors among the training data. This approach is used with the object `sklearn.svm.NuSVC` in the module `scikit-learn`.

Extensions to multi-class case

In case the output variable Y has more than two modalities, there are several ways to extend the methods of the binary case directly.

"One Vs One". In the case where it is desired to predict a label that has $K \ge 3$ classes, we can consider all the pairs of labels (k, l) possible, $1 \le k < l \le K$ (there are $K(K-1)/2$) and learning a classifier $C_{k,l}(X)$ for each of them. The prediction then corresponds to the label that won the most "duels".

"One Vs. All". For each class k, we learn a classifier to discriminate between populations $Y = k$ and $Y \ne k$. From the a posteriori probability estimates, we assign the most probable estimate.

- Implementation -

We will use the object `sklearn.svm.SVC` :

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

The file `svm_script.py` provides a complete example of classification using the function `SVC`. Use this example to familiarize yourself with the syntax and then reproduce a similar experience with the data `iris` as suggested below.

1) Based on the documentation at the following link :

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

   Write a code that will classify class 1 against class 2 of the dataset `iris` using the first two variables and a linear kernel. Leaving half of the data aside, evaluate the model's generalization performance. The dataset `iris` is obtained using the following lines of code :

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

2) Compare the result with the one obtained by using a polynomial kernel-based SVM.
3) Show that the primal problem solved by the SVM can be rewritten as follows :

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

4) Explain the sentence : "An SVM minimizes the classification error by using a convex majorant of the function which is 1 when the margin is negative and 0 otherwise." The function $x \to [1-x]_+ = \max(0, 1-x)$ is called *Hinge* (charnière in French).

SVM GUI

- Start the script `svm_gui.py` available at the link : http://scikit-learn.org/stable/auto_examples/applications/svm_gui.html
  This application allows a real-time evaluation of the impact that the choice of the kernel and the regularization parameter C have.
- Generate a very unbalanced data set with much more points in one class than in the other (at least 90% vs 10%).
- Using a linear kernel and decreasing the parameter C what do you observe ?
  This phenomenon can be corrected in practice by weighting more errors on the lesser class (parameter `class_weight` de SVC) or by a re-calibration technique (used with `SVC(..., probability=True)`).

Classification of faces

The following example is a face classification problem. The database to be used is available at the following address : http://perso.telecom-paristech.fr/~gramfort/lfw.zip .

By modifying the sample code given in the last part of the file `svm_script.py` :

5) Show the influence of the regularization parameter. For example, the prediction error can be displayed as a function of C on a logarithmic scale between `1e5` and `1e-5`.
6) By adding nuisance variables, thus increasing the number of variables to the number of learning points fixed, show that performance drops.
7) Explain why the features are centered and reduced `l.148-150`.
8) What is the effect of choosing a non-linear RBF kernel on prediction ? You will be able to improve the prediction with a reduction of dimension based on the object `sklearn.decomposition.RandomizedPCA`.

To go further : Computation of the duality gap

The following example :

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

explains how to access the parameters estimated by the model (vector of coefficients $\mathbf{w}$ in the attribute `coef_`, w_0 attribute `intercept`, list of support vectors, coefficients of the dual problem).

- Based on this example, write a code that will calculate the value of the primal and dual objective function values and verify that the values are close.
- How does the difference between the two values vary when the optimization tolerance is varied (parameter `tol` of `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
```

```python
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