# Lab 3 SVM

January 20, 2017

Some website :

- `http://scikit-learn.org/stable/modules/svm.html`
- `http://en.wikipedia.org/wiki/Support_vector_machine`

This lab aims at introducing SVM and use them on real and synthetic datas using the package scikit-learn. We first recall some definitions and notations

## 1 Introduction and theoritical background

- We denote $\mathcal{Y}$ the labels. Usually $\mathcal{Y} = \{-1, +1\}$
- $x = (x_1, \cdots, x_p) \in \mathcal{X} \subset \mathbb{R}^p$ is an observation
- $Dn = \{(x_i, y_i), i = 1, \cdots, n\}$ a training set containing $n$ examples and their labels
- We assume that there exists a probabilistic model explaining the generation of the datas from random variables $X$ and $Y$ :

$$\forall i \in \{1, \cdots, n\},\ (x_i, y_i) \overset{(i.i.d.)}{\sim} (X, Y)$$

- We aim at defining from the training set $\mathcal{D}_n$ a function $\widehat{f} : \mathcal{X} \to \{-1, 1\}$ predicting for any new point $x$ its label. Here the decision rule will be linear, that is we separate the space by an hyperplane and we predict $-1$ or $1$ according to the position of $x$ with respect to this hyperplane.

**SVM and kernel for binary classification**
Non linear SVM involve an implicit function $\Phi$ transforming the input space $\mathcal{X} \subset \mathbb{R}^p$ into an Hilbert space $(\mathcal{H}, <.,.>$ whose dimension is greater. Learning is done from the model $(\Phi(X), Y)$ in $\mathcal{H}$. What is expected is that in this new feature space, the data become more linearly separable. From the practical point of view let us observe that one does not compute the projections of $\Phi(X)$. We only use the scalar products $< \Phi(X), \Phi(X') >$. These scalar products can be computed using a kernel (kernel trick)

$$K(x, x') =< \Phi(x), \Phi(x') >$$

We need to choose carefully a kernel. usual choices are the following

- Linear kernel : $K(x, x') =< x, x' >$ (corresponding to linear SVM)

- Radial Gaussian kernel (Gaussian RBF)

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

- Polynomial kernels $K(x, x') = (\alpha + \beta < x, x' >)^\delta$
- Laplace radial kernel $K(x, x') = \exp(-\gamma \|x - x'\|)$
- Sigmoide kernel $K(x, x') = tanh(\alpha + \beta < x, x' >)$

A SVM classifier is of the form

$$\widehat{f}_{w,w_0}(x) = sign(< w, \Phi(x) > + w_0)$$

where $w \in \mathcal{H}, w_0 \in \mathbb{R}$ are parameters learning from the training set $\mathcal{D}_n = \{(x_i, y_i), i \in \{1, \cdots, n\}\}$. The boundary associated to this decsion rule is the set $\{x, < w, \Phi(x) > + w_0 = 0\}$. This et corresponds to an hyperplane in $\mathcal{H}$ but is much more complex in $\mathcal{X}$. In $\mathcal{H}$, we obtain the separating hyperplane maximizing the margin separating the two classes, that is solving the following optimization problem :

$$\begin{cases} (w^*, w_0^*, \xi^*) \in \text{argmin}_{w \in \mathcal{H}, w_0 \in \mathbb{R}, \xi \in \mathbb{R}^n} \left( \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \right) \\ \text{s.c. } \xi_i \geq 0 \\ y_i(< w, \Phi(x_i) > + w_0) \geq 1 - \xi_i \end{cases}$$

The solution $w$ of this problem is of the form

$$w = \sum_{i=1}^n \alpha_i^* y_i \Phi(x_i)$$

The coefficients $\alpha_i^*$ are solutions of the dual problem

$$\begin{cases} \alpha^* \in \text{argmax}_{\alpha \in \mathbb{R}^n} \left( \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{1 \leq i,j \leq n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right) \\ \text{s.c. } 0 \leq \alpha \leq C, \forall i \in \{1, \cdots, n\} \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases}$$

The parameter $C$ is related to the complexity of the classifier. More precisely, it measures the cost of a misclassification. The larger is $C$, the more complex is the decision rule. This approach is called the $C$ classification. One uses the object `sklearn.svm.SVC` in `scikit-learn`. Another approach consists in considering the following alternative problem :

$$\begin{cases} \alpha^* \in \text{argmax}_{\alpha \in \mathbb{R}^n} \left( \sum_{1 \leq i,j \leq n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right) \\ \text{s.c. } 0 \leq \alpha \leq 1, \forall i \in \{1, \cdots, n\} \\ \sum_{i=1}^n \alpha_i y_i = 0 \sum_{i=1}^n \alpha_i \geq \nu \end{cases}$$

where $\nu \in [0, 1]$. This approach is used with the object `sklearn.svm.NuSVC` of the package `scikit-learn`.

**Extensions to the multiclass setting**

In the case where the output variable $Y$ can take more than two values there is several ways to extend the methods of the binary case

- **"One against one"** In the case where we want to predict a label taking more than two values one can consider all the possible pairs of label $(k, l)$ and fit a classifier $C_{k,l}$ for each one. One then predicts the label who won most of the comparisons
- **"One against all"** For each possible value of the output variable, we learn a classifier allowing to discriminate between the two populations $Y = k$ and $Y \neq k$. Using the a posteriori probabilities, one affects the most probable label.

## 2 SVM in practice

We shall use the object `sklearn.svm.SVC` :
`from sklearn.svm import SVC`

1. Use the website :
   `http://scikit-learn.org/stable/modules/svm.html`
   and the dataset `Iris`. Implement a classifier which classifies class 1 against class 2 of the dataset `iris` using the two first variabme and a linear kernel. Use half of the dataset for training and half of the dataset for validation. To import the dataset `iris`, type
   ```
   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 a SVM based on polynomial kernel
3. Prove that the primal problem can also be reformulated as follows

$$\text{argmin}_{w\in\mathcal{H},w_0\in\mathbb{R}}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}[1 - y_i(< w, \Phi(x_i) > +w_0)]_+$$

4. Use the script `svm_gui.py` avaailable on the following website : `http://scikit-learn.org/stable/auto_e`
   This application allows to evaluate the impact of the choice of the kernel and the regularisation parameter $C$.
5. Generate a dataset with much more observations in one class than another (for e.g. 90% vs 10%).
6. Use a linear kernel and decrease the parameter $C$. Comment

**A practical exemple : faces classification**
The following example is a faces classification problem. The dataset is available on the following website
`http://perso.telecom-paristech.fr/ gramfort/lfw.zip`

1. Show the influence of the regularisation parameter
2. Explain why the features are centered and reduced

3. What is the effect of the choice of a non linear kernel RBF on prediction? You may improve the prediction properties using a dimension reduction based on the object `sklearn.decomposition.RandomizedPCA`

**More about the duality gap**

Calcul du saut de dualité The example
`http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#example-svm-plot-separating-hyperplane-py`
explain how access to the parameters of the model ($w$ : `coef_`, $w_0$ `intercept`).

1. Use this example to implement a code calculating the value of the primal and dual functional. Check that these two values are close

2. How does the difference between the two vales vary when the parameter `tol` of SVC vary?

```
==================================================
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()
```