Fundamentals of Machine Learning

Support Vector Machines, Practical Session

Master DSC/MLDM/3DMTT

Assignment: Due date 23/03/2018 22:00 on Claroline Connect
Be careful the deadline is hard! - It is recommended to post your work (zip or tar.gz file with a report in pdf) as soon as possible - this work can be done in groups of 2 students.

Note: all the material is available on the claroline connect page!

The goal of this practical is to use the SVM implementation of the scikit-learn library.

1 Little Warm-Up

To familiarize yourself with SVM behavior, you can have a look to the following demos on the Web:

- http://cs.stanford.edu/people/karpathy/svmjs/demo/
- https://www.csie.ntu.edu.tw/~cjlin/libsvm/
  go in the *Graphic Interface* section (there is a Java and a javascript version).

The first demo allows you to compare the effect of RBF kernel and linear kernel (with margin band highlighted). The second to compare different kernels, you can type the following options in the option bar:

- `-t <value>`: specify the kernel function (0 for the linear kernel, 1 for the polynomial one, 2 for the RBF one, 3 for the sigmoid one),
- `-c <value>`: value of the C parameter,
- `-g <value>`: value of $1/\sigma^2$ in the Gaussian/RBF kernel,
- `-d <value>`: degree of the polynomial kernel,
- `-r <value>`: value of additional parameter in the kernel function (check the document just below the graphic).

You can try to compare different settings with these demos. Note that `libsvm` is a SVM software that implements various tool around SVM.
(You do not have to report your work on this part in your assignment)

2 SVM with Scikit-Learn

The documentation of the Scikit-learn library is available at this url: http://scikit-learn.org/stable/modules/svm.html

We will use mainly the SVC class, the documentation can be found here, the implementation is mainly based on the libSVM library.:
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
You will need mainly the following options:

- **C**: penalty parameter of the error term, default 1.0

- **kernel**: default 'rbf', options: linear', 'poly', 'rbf', 'sigmoid', 'precomputed'
- **gamma**: option for rbf kernel, default $1/n\_features$
- **degree**: for polynomial kernel, default 3.
- **coef0**: bias term for polynomial kernel (and sigmoid)
- **decision_function_shape**: for multiclass classification, it uses 'ovr': one versus rest, and 'ovo' one versus one methods.
- field **support_vectors_**: the list of support vectors

Other options/fields exist, like **class_weight** that allows to weight the different classes in unbalanced settings, check the documentation.

A first example:

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]]) #we create 4 examples
y = np.array([-1, -1, 1, 1])
from sklearn.svm import SVC #import de la classe SVC pour SVM
classif=SVC() #we create a SVM with default parameters
classif.fit(X,y) #we learn the model according to given data
res=classif.predict([[-0.8, -1]]) #prediction on a new sample
print(res);
```

We will now create a 2d graphic to illustrate the learning result

```
import matplotlib.pyplot as plt #the library for plotting
#we create a mesh to plot in
h = .02  # grid step
x_min= X[:, 0].min() - 1
x_max= X[:, 0].max() + 1
y_min = X[:, 1].min() - 1
y_max = X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
#the grid is created, the intersections are in xx and yy

mysvc= SVC(kernel='linear', C = 2.0)
mysvc.fit(X,y)
Z2d = mysvc.predict(np.c_[xx.ravel(),yy.ravel()]) # we predict all the grid
Z2d=Z2d.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx,yy,Z2d, cmap=plt.cm.Paired)
# We plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
plt.show()
```

Note some informations about the colors can be found here http://matplotlib.org/examples/color/colormaps_reference.html.

Work to do:

- Try to create different settings by hand and compare the behavior of linear and non linear classifiers by using different kernels and parameters

3 Dataset Generation

3.1 Random datasets

You can generate automatically some datasets thanks to the function `make_classification`. Here is an example for generating a datasets with 2 features:

```
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=50,n_features=2, n_redundant=0, n_informative=2,
                           random_state=2, n_clusters_per_class=1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
plt.show()
```

Arguments are relatively explicit, you can anyway find more information here
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

Work to do:

- Create a dataset and implement a cross-validation procedure to select the best hyperparameters for a linear classifier and a rbf-kernel classifier (you can also test a polynomial kernel if you have time).
- Display the decision surface found (note that you can save a plot with the instruction `save`).
- Try to display the support vectors on the same plot.

To implement the cross-validation procedure, you can use the same procedure as in the 1st practical. Note there exists also the function `cross_val_score` that can automate the cross-validation procedure (the number of folds is 3 by default), more information here. http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score
There is also a grid search function:
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

3.2 Existing datasets

Apply the same procedure as in the previous section to 2 existing datasets.

- The 2-moons dataset provided in the scikit-learn distribution

  ```
  from sklearn.datasets import make_moons
  X, y = make_moons(noise = 0.1, random_state=1, n_samples=40)
  plt.scatter(X[:,0],X[:,1], c = y, s = 100)
  ```

- The iris dataset used in the 1st practical session, note that this is a multiclass dataset. To be able to display the results, you can take into account only the last 2 attributes.

4 'Real' dataset

Download the ozon dataset on claroline (`ozone.dat`). This dataset comes from MétéoFrance records and is constituted of the following attributes:

- `JOUR`: type of day, holiday (1) or not (0).
- `O3obs`: ozone concentration observed the next day at 17h (local time), often at the maximum pollution rate.
- `MOCAGE`: prediction of the pollution made by a deterministic model/
- `TEMPE`: temperature for the next day at 17h.

- `RMH2O`: humidity rate
- `NO2`: nitrogen dioxide concentration
- `NO`: nitric oxide concentration
- `STATION`: place from where the observations are taken (Aix-en-Provence, Rambouillet, Munchhausen, Cadarache or Plan de Cuques)
- `VentMOD`: wind strength
- `VentANG`: wind orientation

A first remark is that the STATION attribute contains categorical values that have to be converted. A first solution would consist in associating an integer to each station, however this is not very relevant: it biases the dataset because some stations will get more "importance" than others just because of the integer choice. A better solution is to replace each value by a binary vector containing only one 1. Example `Aix-en-Provence` could be encoded as $1, 0, 0, 0, 0$, `Rambouillet` as $0, 1, 0, 0, 0$, `Munchhausen` as $0, 0, 1, 0, 0$ and so on.

The dataset is in CSV like format (with space used as a separator). To load the dataset you can use the `read_csv` function from `panda` library.

```
import pandas as pd
input_file = "mydata.csv"
df = pd.read_csv(input_file)
```

The delimiter can be chose for example, more information here:
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Another point is data scaling: if you do not scale data, attributes defined over large intervals will take more importance than those defined on a significantly smaller interval. A way to tackle this problem is to use a normalizing function

```
from sklearn.preprocessing import StandardScaler
...
# normalize dataset for easier parameter selection
X = StandardScaler().fit_transform(X) #where X is your data matrix
```

Work to do: classification problem by predicting ozone peaks

- We define an ozone peak as an `O3obs` value greater than 150. Assuming that the `O3obs` value (non normalized) is located in the first column of a matrix Z, then you can use the following commande to create your classes:
  `y = Z[:,0] > 150`
- You must then select the correct attributes from the CSV file to define the data matrix and the label set.
- To evaluate correctly the models, you must use a cross-validation procedure. However, to tune the hyperparameters you must apply another cross-validation on the training dataset and then learn the model on the full training data one the parameters have been chosen.
- You must compare different svm classifiers. What is the best performance with cross validation that you can achieve on this dataset, with or without data normalization.

Bonus - Regression problem: predicting the ozone concentration Try now to cast the problem into a regression problem where the objective to predict the value of the ozone peak.
You can use Support Vector Regression (SVR) more information here:
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR

Note that this is a bonus question.