Instructions for Feature Selection Lab: PASCAL Bootcamp 2007

Amir Saffari
Institute for Computer Graphics and Vision
Graz University of Technology, Graz, Austria.
`amir@ymer.org`

Isabelle Guyon
Clopinet Consulting, 955 Creston road
Berkeley, CA 94708, USA.
`isabelle@clopinet.com`

July 1, 2007

1 Introduction

Welcome to the first practical session of the feature selection lab of PASCAL Bootcamp 2007. In this lab, you will be introduced to the Spider and Clop machine learning toolboxes. We will start with very simple problems and try to warm up a bit towards more complicated problems which you will be working later during the bootcamp.

2 Installation Guide

The programs used in this lab are mainly based on MATLAB, as a result you will need to have a running MATLAB on your computer/laptop. If you happened not to have one, do not worry, you will be provided with desktop computers with MATLAB included. So now, if you do not have a MATLAB on your personal computer, you can skip to the next section, otherwise, keep reading to get to know what you should do to install necessary programs.

Please go to this address and download the CLOP package (93 MB):
`http://clopinet.com/isabelle/Projects/NIPS2003/DataNcode.zip`

Now decompress the `DataNcode.zip` into a directory of your choice, we will use the `path2CLOP` to refer to the location where you extracted the `DataNcode.zip`. After extraction, you should get a directory named `DataNcode` inside `path2CLOP` path.

If your operating system is a 32-bit Windows, then you can skip to the next part, else if you are using Linux, then we need to compile a package here. The `svc` model is originally based on a C code, so depending on your machines configuration, there might be a need for compilation. We have provided pre-compiled versions for 32-bit Windows and they usually run without problems. But for Linux, you need to compile them. The source code for `svc` is located in `path2CLOP/DataNcode/Clop/challenge_objects/packages/libsvm-mat-2.8-1`.

Before proceeding, make sure you have *GNU GCC* and *Make* installed on your machine. Please open a terminal and change to the above path. There are two different Makefiles: `Makefile_orig` is the one which was provided by the authors of the SVM package, and `Makefile_amir` is what I used on my machine to compile it. The only difference in these two files is the name of the C compiler. In terminal, please run:

```
$ cd path2CLOP/DataNcode/Clop/challenge_objects/packages/libsvm-mat-2.8-1
$ make -f Makefile_amir
```

This will hopefully compile this package for you[1].

Great, you have finished the installation. Let's check if everything is working.

2.1 Testing the installation

Please start the MATLAB program, and change the working directory to `path2CLOP/DataNcode`, and run the `simpleMain.m` script (replace the `path2CLOP` with yours):

```
>> cd path2CLOP/DataNcode
>> simpleMain
```

If everything is OK, you should get a few lines indicating the license for Spider and CLOP packages, and a few messages showing that the program is loading the `Arcene` dataset, and then finally a support vector machine classifier will be trained on this dataset. If you get any error during these operations, please contact with us, we will try to fix the problem.

3 Getting hands dirty!

Now that we made sure the Clop and Spider are working properly, it is time to start doing some interesting experiments. Please start MATLAB, and change the working directory to `path2CLOP/DataNcode`:

```
>> cd path2CLOP/DataNcode
```

Now we have to add the Clop and its subdirectories to MATLAB search path:

```
>> addpath( genpath( pwd ) )
```

Ok, let's load a dataset which is generated from samples drawn from a combination of two Gaussian distributions:

```
>> load demoData1
>> whos
```

Let's look at the dataset and plot the training data:

[1] I was not able to get the svc running under 64-bit Linux and GCC, it seems that it needs a few code tunings. If you could, please let me know.

```
>> demoData
>> demoData.train
>> figure
>> posClassIndex  = demoData.train.Y > 0;
>> negClassIndex  = demoData.train.Y < 0;
>> plot( demoData.train.X( posClassIndex , 1) , demoData.train.X( posClassIndex , 2) , '.' )
>> hold on
>> plot( demoData.train.X( negClassIndex , 1) , demoData.train.X( negClassIndex , 2) , 'r.' )
>> hold off
```

Alright, let's do it, bring the support vector machine on:

```
>> myModel = svc( { 'coef0=0' , 'degree=1' , 'gamma=0' , 'shrinkage=1' } )
>> [ trainOutput , myModel ] = train( myModel , demoData.train )
>> BER   = balanced_errate( trainOutput.X , trainOutput.Y )
>> EBAR  = error_bar( BER , length( find( trainOutput.Y > 0 ) ) )
```

Let's look at what happened:

```
>> figure
>> posClassIndex  = trainOutput.X > 0;
>> negClassIndex  = trainOutput.X < 0;
>> plot( demoData.train.X( posClassIndex , 1) , demoData.train.X( posClassIndex , 2) , '.' )
>> hold on
>> plot( demoData.train.X( negClassIndex , 1) , demoData.train.X( negClassIndex , 2) , 'r.' )
>> hold off
>> title( 'Training results' )
```

Well, I think we need to test it too:

```
>> testOutput  = test( myModel , demoData.test );
>> BER   = balanced_errate( testOutput.X , testOutput.Y )
>> EBAR  = error_bar( BER , length( find( testOutput.Y > 0 ) ) )
>> figure
>> posClassIndex  = testOutput.X > 0;
>> negClassIndex  = testOutput.X < 0;
>> plot( demoData.test.X( posClassIndex , 1) , demoData.test.X( posClassIndex , 2) , '.' )
>> hold on
>> plot( demoData.test.X( negClassIndex , 1) , demoData.test.X( negClassIndex , 2) , 'r.' )
>> hold off
>> title( 'Test results' )
```

What will happen if I would use Naive Bayes: simple, change `myModel` to be a `naive`, and do what came afterwards, but it is tedius, I know, so just open an m-file and copy all those commands into it. Now we can just edit this file whenever we want to experiment something new. Alright, I know you are lazy to do that, just open the demo1.m file, and there you go.