Learning and Perception practice, 2005

Practice 0

The work environment:
Basic experiments with the Iris corpus

Facultad de Informática,
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

1. Introduction

The goal of this practice is to use some of the most basic tools of the GNU-Linux: “bash” shell, string processing program “awk”, graphic representation “gnuplot” and image viewer “xy”. It is recommended to read the appendix A-D for those that do not know these tools. For more information:

- man bash awk gnuplot xv; gv $(locate gnuplot.ps); gv $(locate xvdocs.ps)
- http://www.faqs.org/docs/Linux-HOWTO/Bash-Prog-Intro-HOWTO.html
- http://grulla.hispalinux.es/articles/scripting
- http://inicia.es/de/chube/Manual_Awk/Menus.htm
- http://www.gnuplot.info/docs/gnuplot.html

For the next practice these tools will help us to perform the experiments and to build specific pattern recognition applications using small generic programs that perform preprocessing, learning and classification.

For this practice it is proposed to use these basic tools for performing small pattern recognition experiments with a small corpus of the pattern recognition field: Iris.

The Iris problem is an academic problem proposed by Fisher in 1936. It consists in the classification of flowers of the iris family using their petal and sepal sizes.

The data set contains the measures of 150 specimens of three subclasses: Setosa, Versicolor and Virginica. It is frequently used as an example task for comparing the performance and possibilities of different methods of data analysis and pattern recognition.
2. The "Iris" data

It is in the directory:
"/labos/asignaturas/fi/app/prac1/iris".
It is a small corpus, so we can copy to our home directory:
"cp -r /labos/asignaturas/fi/app/prac1/iris ."
Alternatively, environment variables can be defined to avoid to write each time the complete path of the files:
```
export APP=/labos/asignaturas/fi/app
export IRIS=$APP/iris
```

Now we can do: "echo $APP; echo $IRIS" or, for instance, "ls -l $IRIS".
These environment variables should be included in the initialization file for the bash shell (.bash_profile or /.profile). By this way we do not need to declare these variables for each new session.

In $IRIS there are two important files: iris.names, is the information of this corpus, and iris.data contains the small corpus. We can see this corpus file using less, more or cat:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa

7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor

6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
```

Each row has a 4-dimensional vector and one label at the end. To simplify the posterior processes we need to represent each component of the vector and the class-label separated by one space instead of commas, and the class-label should be simplify by removing the redundant part ("Iris-"). This can be easily achieve by the next awk program:

```
cat $IRIS/iris.data |
LANG=C awk -F " \" '{
    for (i=1;i<NF;i++) printf(" %4.1f", $i);
    printf(" %s\n", substr($NF,6))
}
```

And we obtain:
```
5.1  3.5  1.4  0.2  setosa
4.9  3.0  1.4  0.2  setosa
4.7  3.2  1.3  0.2  setosa

7.0  3.2  4.7  1.4  versicolor
6.4  3.2  4.5  1.5  versicolor
6.9  3.1  4.9  1.5  versicolor
```
6.3 3.3 6.0 2.5 virginica  
5.8 2.7 5.1 1.9 virginica  
7.1 3.0 5.9 2.1 virginica

It is interesting to save this command in a *shell-script* file. Then, copy the text of this command to a file, name it, for example: “limpiaIris” and save it. In this file we should include the first line: ‘#!/bin/bash’ to guarantee that the file is executed using the bash shell. Finally, we need to give execute permissions to this file: “chmod +x limpiaIris”.

3. Graphic representation

We can plot the data using the *gnuplot* program.

Initially we need to obtain bi-dimensional plots using all the pairs of the data vector components, 4 components for the iris data, for instance the pair “petal height”-“sepal width” (components 3 and 4 respectively).

The gnuplot can represent in the same plot several data sets, each data set with different symbols. Then first, we should split the data saving each class in different files (*setosa, versicolor, virginica*):

```bash
for c in setosa versicolor virginica ; do
    limpiaIris | grep $c > ${c}.txt
done
```

Each file only contains the components of those vectors belonging to the class associated to this file.

A more efficient way to split the data in different class-files is using the awk:

```bash
./limpiaIris | awk '{print >${NF}.txt}'
```

This command could be integrated into the *limpiaIris* script.

Now we can use *gnuplot* to plot the three classes. First run gnuplot (we enter in the interactive mode), then type:

```bash
plot 'setosa.txt' u 3:4, 'versicolor.txt' u 3:4, 'virginica.txt' u 3:4
```

or:

```bash
plot 'setosa.txt' u 3:4  
replot 'versicolor.txt' u 3:4  
replot 'virginica.txt' u 3:4
```

You can exit from gnuplot using Ctrl+D or the “quit” command.

Now, we do the same but using an non-interactive mode:

```bash
echo "plot 'setosa.txt' u 3:4, 'versicolor.txt' u 3:4, 'virginica.txt' u 3:4" | gnuplot -persist
```
To improve the graphic representation we can select a different range for the axes:

\[
\text{set xrange [0.5:7.5] ; set yrange [-0.25:2.75] ; replot}
\]

Also, it is necessary to titled the axes and the graphic:

\[
\text{set encoding iso_8859_1}
\text{set xlabel 'height' ; set ylabel 'width'}
\text{set title 'Three iris flowers classes using the sepal and petal sizes'}
\text{replot}
\]

Generally, once the graphic is correct we would like to save it. We can save all the gnuplot command by using the command “save”: “\text{save ‘irisPetalos.gnp’}”. Also, we would like to produce the graphic using some standard format, for instance:

\[
\text{set terminal postscript color solid}
\text{set output ‘irisPetalos.ps’}
\]

The PostScript file obtained can be visualized using the “gv”program.

We can do all using the non-interactive mode:

\[
\text{echo "set encoding iso_8859_1}
\text{set xlabel 'longitud' ; set ylabel 'anchura'}
\text{set title 'Tres clases de flores iris según el tamaño de sus pétalos'}
\text{set terminal postscript color solid}
\text{set output ‘irisPetalos.ps’}
\text{set xrange [0.5:7.5]}
\text{set yrange [-0.25:2.75]}
\text{set key left}
\text{set encoding iso_8859_1}
\text{plot 'setosa.txt' u 3:4,\
   'versicolor.txt' u 3:4,\
   'virginica.txt' u 3:4" |}
\text{gnuplot}
\]

The resulting graphic is like this:
4. A simple classification experiment

It is possible to perform pattern recognition experiments using the bash and awk programs. Now we learn how to perform a simple experiment with the iris data. We are going to implement the "Nearest Neighbour" (NN) method. First, we split the data into two different subsets A and B. We use the subset A, to learn (training set), while we use the subset B, to measure the accuracy of our classifier.

The learning process for the NN method is straightforward: just storing in memory all the vectors of the training data (prototypes) and the class-label associated to each one.

The classification consists in given a test vector, \( \mathbf{y} \in \mathcal{B} \), and \( \mathbf{p} \in \mathcal{A} \) the nearest prototype. The NN method assigns to \( \mathbf{y} \) the class associated to \( \mathbf{p} \). Since we have the “correct” class \( c' \) of all \( \mathbf{y} \in \mathcal{A} \), we can measure the accuracy of the classifier. If \( c = c' \) the NN classifier had correctly classified the object. To measure the distance between \( \mathbf{y} \) and the prototypes of \( \mathcal{A} \) we need to use some suitable measure. In this case we use the Euclidean distance.

The next script split the data into two subsets with the 50% of the data for training (A) and the other 50% for testing (B):

```
# Dividimos cada clase del corpus en entrenamiento (A) y test (B):
wc -l *.txt
rm -f irisA.txt irisB.txt
for c in setosa versicolor virginica ; do
echo $c
    head -25 ${c}.txt >> irisA.txt ; tail -25 ${c}.txt >> irisB.txt
done
wc -l irisA.txt irisB.txt
```

Now we perform the classification of the prototypes in B:

```
cat irisB.txt | LANG=C awk -v A=irisA.txt
   BEGIN{# Ponemos el conjunto de entrenamiento en el array asociativo M
           while ((getline <A)) {c=$NF; NF--; M[$0]=c; /* print $0,M[$0] */}
              errores=0
              printf("Clases: correcta sistema Errores\n\n")
        }# Ahora procesamos los vectores de test que llegan de la entrada std
        cc=$NF; NF--; dMin=999999999.9; cBest=""
        split($0,y) # y es un array con los componentes del vector de entrada
        for (p in M) {
           c=M[p]; split(p,v); # v es un array con los componentes del vector p
           d = 0; for (i=1;i<=4;i++) d+= (y[i]-v[i])*(y[i]-v[i]);
           if (d<dMin) {dMin=d; cBest=c}
        }
        if (cc != cBest) errores++
        printf("%17s %12s %2d\n", cc, cBest, errores)
    }END{
        printf("\nTotal: %d datos, %d errores (%.2f\%)\n", NR, errores, 100*errores/NR)
    }
```

This program generates this output:
<table>
<thead>
<tr>
<th>Clases: correcta sistema</th>
<th>Errores</th>
</tr>
</thead>
<tbody>
<tr>
<td>setosa setosa</td>
<td>0</td>
</tr>
<tr>
<td>setosa setosa</td>
<td>0</td>
</tr>
<tr>
<td>versicolor versicolor</td>
<td>0</td>
</tr>
<tr>
<td>versicolor versicolor</td>
<td>0</td>
</tr>
<tr>
<td>versicolor versicolor</td>
<td>0</td>
</tr>
<tr>
<td>versicolor versicolor</td>
<td>0</td>
</tr>
<tr>
<td>versicolor virginica</td>
<td>1</td>
</tr>
<tr>
<td>versicolor versicolor</td>
<td>1</td>
</tr>
<tr>
<td>virginica virginica</td>
<td>1</td>
</tr>
<tr>
<td>virginica versicolor</td>
<td>2</td>
</tr>
<tr>
<td>virginica virginica</td>
<td>2</td>
</tr>
<tr>
<td>virginica versicolor</td>
<td>3</td>
</tr>
<tr>
<td>virginica virginica</td>
<td>3</td>
</tr>
<tr>
<td>virginica versicolor</td>
<td>4</td>
</tr>
<tr>
<td>virginica virginica</td>
<td>4</td>
</tr>
</tbody>
</table>

Total: 75 datos, 4 errores (5.33%)