Project report: Machine Learning

Giovanni Simoni
Register 142955
giovanni.simoni@roundhousecode.com

September 8, 2011

Contents

1 Overview **2**

1.1 Objective 2

1.2 The dataset 2

1.3 Computational steps 2

2 Approaching the problem **2**

2.1 Recognizing Thresholds 3

2.1.1 Entropy of the Dataset 3

2.1.2 Computing the threshold 3

2.1.3 Ranking the genes 4

2.2 Clustering of Genes 4

2.2.1 Distance between clusters 4

2.2.2 Translating genes into vectors 5

3 The programs **5**

3.1 Usage 5

3.2 Structure 7

4 Experimental results **7**

1 Overview

1.1 Objective

The project aimed to achieve a clustering among a given set of genes, basing on their expression level. I was provided with a dataset of 72 records for patients affected either by *Acute Lymphoblastic Leukemia* or by *Acute Myeloid Leukemia*. Each record of the given dataset reports the expression levels for 5147 genes.

For some genes, the expression level is a good index for discrimination of the two forms of leukemia: basing on how the dataset is partitioned (more details on this later) genes can be ranked according to their precision in splitting the dataset.

My final objective was the hierarchical clustering of the genes having the best splitting performances, based on the similarity in the genes behavior.

1.2 The dataset

The provided dataset is a plain text file with the following format:

- A row containing a *tab-separated* list of genes names;
- A row containing, for each gene, the kind of data represented for it. In this case each feature is *continue* (i.e. a real number);
- An empty row;
- A row for each of the analyzed patients, having a *tab-separated* list of expression levels one for each of the genes listed in the first row of the file. The last column of each patient reports the form of leukemia the patient is affected by.

As we are considering two forms of leukemia, we are in a binary classification setting.

1.3 Computational steps

The available data have been elaborated in two phases:

1. The recognition of the best splitting threshold for each gene (described in Subsection 2.1);
2. The clustering of the more expressive genes basing on how they classify the given dataset (described in Subsection 2.2).

In order to achieve a quick development, the *Python* programming language has been used. I wrote two main programs according to the two elaboration phases required: **genes-rank** and **genes-clust**. The programs are described in Section 3

2 Approaching the problem

In this section the problem will be analyzed from a theoretical perspective.

2.1 Recognizing Thresholds

Some genes can be more related than others with the kind of leukemia the patient is affected by: for a gene having a good correlation, we can infer the classification basing on the expression level, thus classifying differently an instance if the value is above or below a certain threshold.

Supposing we already have a threshold for each gene, since we are in a *supervised learning* setting, we could determine the aforementioned degree of correlation by measuring how good is the threshold-based prediction with respect to the real assignment of examples.

First of all, however, we should determine for each gene the best threshold for the binary classification.

2.1.1 Entropy of the Dataset

If we are given with a certain dataset, which is partitioned according to a set of classes $y_1 \cdots y_c$, we can use the *Information Entropy* as index of uniformity of the dataset.

The Entropy is a measure of the uncertainty of a random variable. For a discrete random variable X taking values $\{ x_1 \cdots x_n \}$ with a certain probability mass function p, the Entropy is defined as:

$$H(X) = \sum_{i=0}^{n} p(x_i) \cdot \log_2 p(x_i)$$

By construction, the Entropy is maximum for a random variable having uniform distribution, while it gets lower values for non-homogeneous distributions.

We can consider a dataset D as a set of values drawn from a categorical distribution, for which we can estimate the probability distribution among the classes $y_1 \cdots y_c$ as the proportion between the classes cardinality $D_1 \cdots D_c$ and the whole set cardinality. In other words we can define:

$$\Pr(x \in y_i) = \frac{|D_i|}{|D|}$$

In our case we need a binary classification, thus examples are partitioned in two subsets, D_{AML} and D_{ALL}, of the dataset D. As we have a mass function we can compute the Entropy of the dataset.

2.1.2 Computing the threshold

The less is the Entropy of a set, the least is the information we gain from knowing an example to belong that set. So far, when splitting a set D into partitions $D_1 \cdots D_N$, we obtain a change in the overall Entropy: this is the concept of Information Gain:

$$IG(D, D_1 \cdots D_N) = H(D) - \sum_{i=0}^{N} \frac{|D_i|}{|D|} H(D_i)$$

For each gene g, we search an expression level threshold t_g to be used as boundary for our dataset partitioning, thus splitting D into two subsets:

$$D_0 = \{ x \in D \mid x_g < t_g \} \qquad D_1 = \{ x \in D \mid x_g \geq t_g \}$$

Once we have a partitioning, we can compute the Information Gain coming from it.

Let $L = \langle \langle x_0, y_0 \rangle \cdots \langle x_m, y_m \rangle \rangle$ be a list of pairs sorted by the first field. Each element of the list corresponds to an element of the dataset, and it's a pair $\langle x, y \rangle$ such that x is the expression level of the gene g and y it's the label of the example. The candidate thresholds are the values

$$T = \left\{ \frac{x_0 + x_1}{2} \;\middle|\; \langle x_0, y_0 \rangle = L_i \wedge \langle x_1, y_1 \rangle = L_{i+1} \wedge y_0 \neq y_1 \right\}$$

The best suited threshold t_g for a gene g is the element of T which maximizes the Information Gain coming from the splitting.

2.1.3 Ranking the genes

As we computed a threshold for each gene, we have as many binary classifiers as the number of analyzed genes. Some of them may have a bad quality, while other may be better than other at fitting the dataset labeling. The ones showing better performances in splitting are likely to be more related with the leukemia type with respect to the other.

The genes can be sorted according to the Information Gain they yield if used as splitting parameter.

2.2 Clustering of Genes

In the *threshold recognition* phase we exploited the examples labels to identify the genes showing a categorization capability with respect to our classes.

In this phase we want to group together genes which show a similar behavior in terms of categorization of examples. In order to do this we rely on the theoretical concepts described in this section.

2.2.1 Distance between clusters

In a n-dimensional vector space $\mathbb{F}^n$, we can give many definitions of *distance*. The most intuitive one is the *euclidean distance*:

$$\text{let } v, w \in \mathbb{F}^n \qquad \text{dist}\,(v, w) = \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2}$$

As we can compute the distance between two vectors, we can also compute the distance between two *groups* of vectors Let $C = \{ c_1 \cdots c_n \}$ and $D = \{ d_1 \cdots d_m \}$ be two clusters we want to compare. The following techniques can be used:

- $\text{dist}\,(C, D) = \text{dist}\,(\text{avg}\,(C), \text{avg}\,(D))$;
- $\text{dist}\,(C, D) = \min \{ \text{dist}\,(x, y) \mid \langle x, y \rangle \in C \times D \}$;
- $\text{dist}\,(C, D) = \max \{ \text{dist}\,(x, y) \mid \langle x, y \rangle \in C \times D \}$;
- $\text{dist}\,(C, D) = \text{avg}\,(\text{dist}\,(x, y) \mid \langle x, y \rangle \in C \times D)$.

The clustering technique consists in an iterative process for which the pair of clusters having the minimum distance gets merged into a single set.

2.2.2 Translating genes into vectors

In the previous phase we managed to define a binary classification threshold for each gene and find the subset G of genes which show a certain correlation with the disease. Now we want to group together genes of G which show a similar way of classifying the dataset.

If we consider a single example x of the dataset, two genes $g_a, g_b \in G$ show the same behavior with respect to x if $x_{g_a} < t_{g_a} \iff x_{g_b} < t_{g_b}$. So far we can define a mapping $f : G \rightarrow \{0,1\}^N$ where N is the number of examples we are provided with.

$$\begin{aligned} \forall g \in G \qquad & f(g) = \langle g_0 \cdots g_N \rangle \in \{0,1\}^N \\ & g_i = 0 \iff x_{i,g} < t_g \end{aligned}$$

Even if a space defined by values in $\{0,1\}$ is not properly a vector space, the *euclidean distance* still works on it, so we can group together similar genes with a clustering technique.

3 The programs

As mentioned in Subsection 1.3, I wrote two different programs, one for each of the phases described in Section 2.

3.1 Usage

The program **genes-rank** requires as parameters the path of the file containing the dataset and a name for the output file:

```
$ ./genes-rank ../dataset_1_2_3 ../results/thresholds.txt
```

After the elaboration phase, it produces as output a text-file containing a list of genes sorted by the maximal information gain, as described in Paragraph 2.1.3. The file produced by **genes-rank** contains rows like the following:

```
Gene 2376 : 154.5 0.698437155782
Gene 1335 : 312.5 0.693746194896
Gene 3544 : 994.0 0.693746194896
Gene 1374 : 1419.5 0.681680475648
...
```

Which can be matched by the *regular expression*

```
Gene (\d+) : (-?\d+\.?\d*) (-?\d+\.?\d*)
```

The first numeric value is the index of the gene with respect to the dataset file, while the second and the third are respectively the threshold and the information gain for that gene.

This output file can be parsed by the **genes-clust** program, which can produce a clustering for genes according to what explained in Subsection 2.2. All the listed distance measure between clusters are available as options for the program:

```
$ ./genes-clust --help
Usage: genes-clust [options] <vectors file>

Available algorithms
        nearest-neighbor
        dist-average
        farthest-neighbor
        average-distance
        all-of-them

Options:
  -h, --help            show this help message and exit
  --algorithm=ALGS      Enable distance algorithm
  --log-file=LOG_FILE   Target file for statistics
  -D DATASET, --dataset=DATASET
                        Dataset file
  -R GENES_RANK, --genes-rank=GENES_RANK
                        Dataset file
  -n TAKEBEST           Number of genes to take
```

The following command, for instance, computes the clustering for the best 100 examples:

```
$ ./genes-clust --algorithm=all-of-them --log-file=log.txt \
      -D ../dataset_1_2_3 -R ../results/thresholds.txt -n 100
```

- All algorithms have been selected, thus the following files will be written:
  - **nearest-neighbor.dot**;
  - **dist-average.dot**;
  - **farthest-neighbor.dot**;
  - **average-distance.dot**.

  Those files are representation in *dot* language of the clustering dendogram produced for each algorithm;
- The file **log.txt** will contain the statistics for all the used algorithms (*sum of squared errors* for each iteration of the clustering mechanism);

Each **.dot** file can be compiled into a *pdf* showing the dendogram:

```
$ for i in *dot; do dot -Tpdf -o $(basename $i .dot).pdf < $i; done
$ ls *pdf
average-distance.pdf  dist-average.pdf  farthest-neighbor.pdf
nearest-neighbor.pdf
```

Figure 1 shows the clustering of the best 10 genes according to the provided dataset.

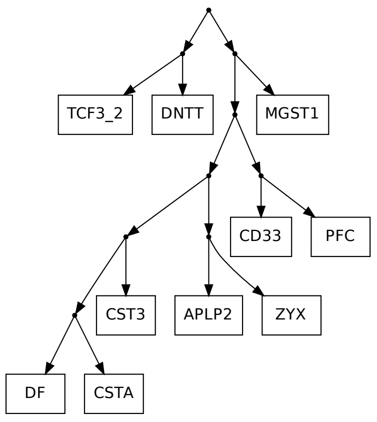

Figure 1: Clustering by *nearest-neighbor* of the best 10 genes

3.2 Structure

Besides the **genes-rank** and **genes-clust** executable files, the program is composed by the following *Python* modules:

- **utils.py**: Generic utilities and algorithms;
- **vecspace.py**: Definition of a basic vector (just to avoid dependencies from external libraries);
- **parser.py**: Parser for the dataset file;
- **dataset.py**: Definition of the problem structure and dataset access structures;
- **thresholds.py**: Genes threshold computation module;
- **hclust.py**: Hierarchical clustering module.

4 Experimental results

For each distance measure and for each iteration of the clustering mechanism, the **genes-clust** program records the *sum of squared errors* inside the log file.

I quckly wrote the **parse-log** program to parse it out and I fed *gnuplot* with the output, producing the graphs shown in Figure 2.

The plot highlights that the quality of the clustering with different distance measure is comparable, although the *farthest neighbor* and the *average distance* techniques seems to work slightly better than the other.

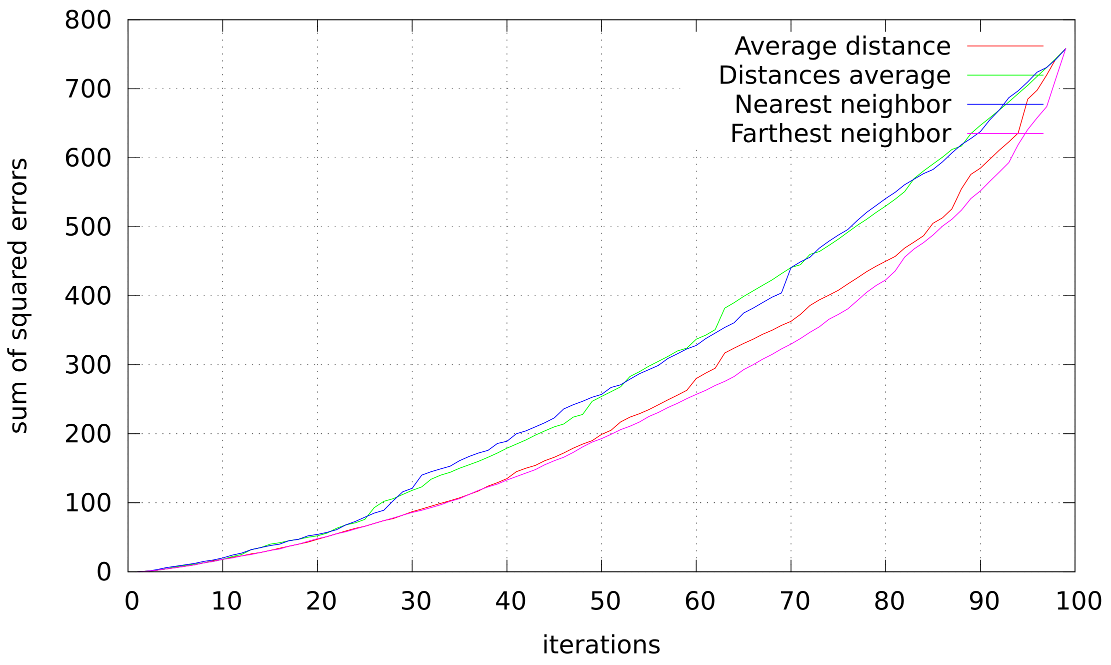

Figure 2: Comparison between clustering quality by using different clustering techniques. The clustering has been run with 100 experiments. Each iteration merges two clusters (thus we have 100 iterations)