BIOINFORMATICS APPLICATIONS NOTE

Vol. 27 no. 12 2011, pages 1729–1730
doi:10.1093/bioinformatics/btr233

Data and text mining

Advance Access publication April 19, 2011

Rgtsp: a generalized top scoring pairs package for class

Vlad Popovici[1,2,*], Eva Budinská[1,3] and Mauro Delorenzi[1]

[1]Bioinformatics Core Facility, Swiss Institute of Bioinformatics, CH-1015 Lausanne, [2]Swiss National Center of Competence in Research Molecular Oncology, School of Life Sciences, Ecole Polytechnique Fédérale de Lausanne, Switzerland and [3]Institute of Biostatistics and Analyses, Masaryk University, Brno, Czech Republic

Associate Editor: Jonathan Wren

ABSTRACT

Summary: A top scoring pair (TSP) classifier consists of a pair of variables whose relative ordering can be used for accurately predicting the class label of a sample. This classification rule has the advantage of being easily interpretable and more robust against technical variations in data, as those due to different microarray platforms. Here we describe a parallel implementation of this classifier which significantly reduces the training time, and a number of extensions, including a multi-class approach, which has the potential of improving the classification performance.

Availability and Implementation: Full `C++` source code and `R` package `Rgtsp` are freely available from http://lausanne.isb-sib.ch/~vpopovic/research/. The implementation relies on existing OpenMP libraries.

Contact: vlad.popovici@isb-sib.ch

Received on November 12, 2010; revised and accepted on April 8, 2011

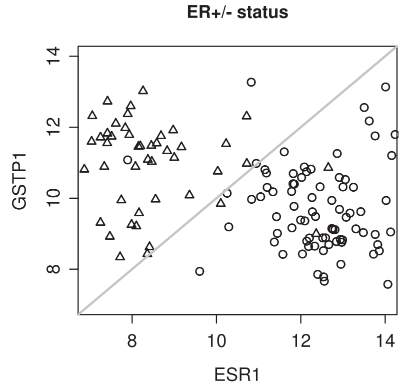

Fig. 1. Predicting estrogen receptor status: if GSTP1 < ESR1, then the sample is considered ER+ (circles), otherwise ER− (triangles).

1 INTRODUCTION

Top scoring pairs (TSPs; Geman *et al.*, 2004) are simple two-variables binary classifiers, in which the prediction of the class label is based solely on the relative ranking of the expression levels of the two genes. The rank-based approach to classification ensures a higher degree of robustness to technical variations and makes the rule easily portable across platforms. Also, the direct comparison of the expression level of the genes is easily interpretable in the clinical context, making the TSPs attractive for medical tests.

Let $\mathbf{x} = [x_i]_{i=1,\ldots,m} \in \mathbb{R}^m$ be a vector of measurements (e.g. gene expression) representing a sample and let the corresponding class label be y, with two classes denoted by 0 and 1. Then, for all pairs of variables i and j, a score is computed,

$$s_{i,j} = P(x_i < x_j | y=1) - P(x_i < x_j | y=0), 1 \leq i,j \leq m \qquad (1)$$

where P are conditional probabilities estimated from the data, and the corresponding decision rule is: if $\mathrm{sign}(s_{i,j})x_i < \mathrm{sign}(s_{i,j})x_j$ then predict $y=1$, otherwise $y=0$. The pairs are ordered by the absolute values of their scores and the top t pairs ($t \geq 1$) are then considered for the final model (Geman *et al.*, 2004; Tan *et al.*, 2005; Xu *et al.*, 2005). Remarkably, training a TSP does not require the optimization of any parameter and does not depend on any threshold. Selecting a suitable value for t should be done following the usual machine learning paradigm for optimizing meta-parameters (see, for example, Hastie *et al.*, 2001). Figure 1 shows an example of a TSP predicting the estrogen receptor status. The decision boundary (in grey) is always a line with a slope of 1.

2 IMPLEMENTATION

While the method briefly described above is simple and poses no implementation problems, using it in the context of highly dimensional data requires the evaluation of an extremely large number of pairs of variables making its usage impractical, especially in the context of resampling techniques for performance estimation. However, most if not all of the modern desktop computers are multi-core machines, making parallel programs a feasible alternative to classical serial ones.

Our implementation in `C++` exploits the multi-core architecture by using the OpenMP libraries of the system (Chapman *et al.*, 2007), and is wrapped in an `R` package – `Rgtsp`. The full source code and the `R` package are available from http://lausanne.isb-sib.ch/~vpopovic/research/. As `C++` is the main implementation language, the library can easily be extended and integrated with other software libraries. Also, the `R` functions are independent of the domain of application so they could be applied to any kind of data.

*To whom correspondence should be addressed.

3 USAGE EXAMPLES

We present a typical case of using `Rgtsp` package. These examples represent solely some code snippets and not the full process of developing and assessing the performance of a classifier.

The data used in these examples consists of 130 samples stage I to III breast cancer (Hess *et al.*, 2006) and the goal is to predict the estrogen receptor status (positive or negative coded with '+1' and '0', respectively). For illustration purposes we use only a subset of full dataset available from GEO repository under accession number GSE16716.

Before starting `R`, the user has the option of choosing the number of processing units that will be used, by setting the environment variable `OMP_NUM_THREADS`. If not set, it defaults to the maximum number of processing units available.

The first steps load the library and the data and build a list of TSPs (note that the matrix *X* contains the variables as columns):

```
> library(Rgtsp)
> data(mdabr)
> tsp.list = tsp.n(X, y.erpos, 500)
> str(tsp.list)
> print(tsp.list)
```

The function `tsp.n()` returns at most *n* TSPs as a list with three components: the first two correspond to the indexes of the selected variables and the third one contains the associated scores. A similar function, `tsp.s()`, returns all the TSPs that have a score larger than a specified value.

For the *p*-th TSP, the prediction rule can be written as: predict class '+1' if `X[,tsp.list$I[p]] < X[,tsp.list$J[p]]` and this forms the core of the `predict` function. The decision function for $p=1$ in the above example is shown in Figure 1. Given a list of TSPs one has different choices on how to obtain the final predicted labels. Currently, `Rgtsp` proposes two means of combining the predictions of individual TSPs: either by majority voting or by weighting the votes with the correspoding scores—giving more weight to the TSPs with better scores. This functionality is available through the `predict()` generic function:

```
> yp = predict(tsp.list, X, combiner="majority")
> sum(yp != y.erpos) # count the errors
[1] 3
```

By inspecting the list of TSPs, it becomes clear that there are variables that are selected many times as having always either higher or lower value than all its pairing variables. We call such a structure a *TSP hub* and we can construct all the hubs larger than a specified size (25 pairs for example) using

```
> h = tsp.hub(tsp.list, min.hub.size=25)
> print(h)
Hub 1: 194 pairs
    Center: 953 >
14 25 42 43 44 45 54 105 140 146 149 150 152 202 ...
```

This corresponds to a TSP hub in which the probeset `colnames(X)[953] (205225_at, ESR1)` has a higher expression than all other probesets in the list `tsp.list`. The TSP hubs can also be used in predicting the labels, through the same mechanism as above:

```
> yph = predict(h, X, combiner="majority")
> sum(yph != y.erpos) # no. of errors: 6
```

We see that in this particular case the prediction by TSP hubs is slightly less accurate than the combined predictions of the individual TSPs.

The generalization performance of the TSPs classifiers can be estimated by various methods. The `Rgtsp` package provides a function for *k*-fold cross-validation of the binary TSP classifiers (either `tsp.n()` or `tsp.s()` functions), `cv.tsp()`, which returns the training and validation performance of the classifier (it defaults to 5-fold cross-validation).

```
> r = cv.tsp(X, y.erpos)
> print(r)
$tr.m
 Error.rate Sensitivity Specificity         AUC
 0.02884615  0.97812500  0.96000000  0.96906250
```

In the case of a multi-class problem, we propose to use classification trees built on top of TSPs predictions. For $C>2$ classes, one can train TSPs to solve each of the $C(C-1)/2$ pairwise binary classification problems [called one-versus-one (Hsu and Lin, 2002) or round robin (Fürnkranz , 2002) strategy] and then combine the predictions of the TSPs through a classification tree to predict the original classes. For more details the reader is referred to the package web page. This approach is implemented in the function `mtsp()` and makes use of the `ctree()` function in the `party R` package (`y4` is an artificial 4–class label vector):

```
> m = mtsp(X, y4)
> yp = predict(m, X)
```

Funding: Swiss National Science Foundation NCCR Molecular Oncology (to V.P. and M.D.); Fondation Medic (to E.B.).

Conflict of Interest: none declared.

REFERENCES

Chapman,B. *et al.* (2007) *Using OpenMP*. The MIT Press.

Fürnkranz,J. (2002) Round robin classification. *J. Mach. Learn. Res.*, **2**, 721–747.

Geman,D. *et al.* (2004) Classifying gene expression profiles from pairwise mRNA comparisons. *Stat. Appl. Genet. Mol. Biol.*, **3**, Article19.

Hastie,T. *et al.* (2001) *The Elements of Statistical Learning. Data Mining, Inference and Prediction*. Springer.

Hess,K.R. *et al.* (2006) Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer. *J. Clin. Oncol.*, **24**, 4236–4244.

Hsu,C.-W. and Lin,C.-J. (2002) A comparison of methods for multiclass support vector machines. *IEEE Trans. Neural Netw.*, **13**, 415–425.

Tan,A.C. *et al.* (2005) Simple decision rules for classifying human cancers from gene expression profiles. *Bioinformatics*, **21**, 3896–3904.

Xu,L. *et al.* (2005) Robust prostate cancer marker genes emerge from direct integration of inter-study microarray data. *Bioinformatics*, **21**, 3905–3911.