Distance Weighted Discrimination and Second Order Cone Programming

Hanwen Huang, Xiaosun Lu,
Yufeng Liu, J. S. Marron, Perry Haaland

October 29, 2011

1 Introduction

This vignette demonstrates the utility and flexibility of the R-package `dwd` in conducting classification and optimization problems. Distance Weighted Discrimination (DWD) is a recently developed powerful classification method which was originally motivated for solving the High Dimensional Low Sample Size (HDLSS) examples (Marron *et al.* (2007)), but can be applied to many other examples as well. One of the big advantages of DWD over Support Vector Machine (SVM) is that it can overcome data piling problem in high dimensional situations. The original DWD paper (Marron *et al.* (2007)) only describes the implementation of the binary classification method. The multiclass version of DWD has also been developed in Huang *et al.* (2011). It is suggested that all users refer to these publications in order to understand the DWD terminology and principles in greater detail.

The implementation of DWD is more challenging than the implementation of SVM because it requires solving an optimization problem called Second Order Cone Programming (SOCP). The existing DWD software was written in Matlab and employed a very efficient SOCP sovler from `SDPT3` (semidefinite-quadratic-linear programming) package developed by Toh *et al.*. `SDPT3` implemented an infeasible path-following algorithm for solving conic optimization problems involving semidefinite, second-order and linear cone constraints.

The R-package `dwd` was developed on the basis of the existing Matlab version but includes some additional features. The key step was to build a R SOCP solver which was implemented using exactlly the same algorithm as used by the corresponding Matlab version. For the convenience of the users who are familiar with using SVM in R, the main classification functions and arguments in `dwd` are formatted in a similar way to the one used by the SVM functions in the `kernlab` package. To help the users in using this software, some examples to illustrate the coding of problem data are provided. In addition, an efficient R Quadratic Programming (QP) solver based on the SOCP is also included in this package which provides a useful tool for those users who want to develop their own SVM package.

2 DWD implementation and output

Both SVM and DWD are margin-based classification methods in the sense that they build the classifier through finding a decision boundary to separate the classes. DWD uses a different criterion from SVM. It seeks to achieve the goal by maximizing the average distance rather than the minimum distance among the classes. Similar to the `ksvm` function from

kernlab, which has been widely used in SVM analysis, we developed a function called kdwd in this package for doing DWD analysis.

To solve multiclass classification problem, two different methods are used. The first one is to use the "one-versus-one" approach, in which classifiers are trained on each pair of classes and the class label is predicted by a voting scheme. The second one is to build a single classifier including all classes simultaneously and solve a big optimization problem.

Every DWD analysis requires two elements from a dataset: (1) a matrix of predictor, which should be in the form of an $n \times d$ matrix, where n represents the sample size and d represents the dimension. (2) a response vector of length n with each element corresponding to one sample. The appropriate scaling steps can be taken by using scaled option. It should be noted that in the current version of dwd, missing values are not allowed, and must be imputed prior to analysis. In this vignette, we will use the iris dataset as illustration.

```
> library(DWD)
> data(iris)
```

iris is a data frame of 4 measurements across 150 samples with response variabe being the species of each sample. The goal of interest is to predict the species based on the given measurement. This is a 3-class classification problem since there are total 3 different species in the samples which are provided in iris$Species.

```
> table(iris$Species)

    setosa versicolor  virginica
        50         50         50
```

An example for the kdwd function is shown below.

```
> results <- kdwd(Species ~ ., data = iris)
> predict(results, iris)
```

The basic output from kdwd is an object of class *kdwd*. Showing objects of class *kdwd* will print details on the results for all classifier included in the model. For each classifier, the optimal solution of the parameters are displayed along with the final primal and dual objective values. The cross validation error rate can also be returned with the argument cross = k, where k is the number of the folds used in the cross-validation.

DWD is different from SVM in that it allows all data points rather than only support vectors to have direct influence on the determination of the decision boundary. So unlike ksvm, the outputs of kdwd do not include any information about the support vectors. The current version of dwd can only solve linear DWD problems. We are still working on incorporating into the kernel trick such that it can be used to solve the general nonlinear problems as well.

More examples of the implementation and the changes to functional arguments of kdwd are detailed in help documents.

3 The Second Order Cone Programming solver

SOCP is a nonlinear convex problem that includes linear and (convex) quadratic programs as special cases. It can be formulated in the following equations:

$$\min \sum_{i=1}^{n_q} \langle c_i^q, x_i^q \rangle + \langle c^l, x^l \rangle + \langle c^u, x^u \rangle$$

$$
\begin{aligned}
s.t. \quad & \sum_{i=1}^{n_q} A_i^q x_i^q + A^l x^l + A^u x^u = b \\
& x_i^q \in K_q^{q_i} \forall i,\ x^l \in K_l^{n_l},\ x^u \in R^{n_u}
\end{aligned}
$$

Here c_i^q, x_i^q are vectors in R^{q_i} and $K_q^{q_i}$ is the quadratic or second-order cone defined by $K_q^{q_i} := \{x = [x_0; \bar{x}] \in R^{q_i} : x_0 \geq \sqrt{\bar{x}^T \bar{x}}\}$. Similarly, c^l, x^l are vectors of dimension n_l, $K_l^{n_l}$ is the nonnegative orthant $R_+^{n_l}$, and c^u, x^u are vectors of dimension n_u. The dual problem associated with the problem above is:

$$
\begin{aligned}
& \min b^T y \\
s.t. \quad & (A_i^q)^T y + z_i^q = c_i^q,\ z_i^q \in K_q^{q_i},\ i = 1, \cdots, n_q \\
& (A^l)^T y + z^l = c^l,\ z^l \in K_l^{n_l}, \\
& (A^u)^T y = c^l,\ y \in R^m.
\end{aligned}
$$

DWD is one of the important applications of SOCP, but SOCP can be used to solve many other problems as well. The package `kdwd` provides a stand-alone SOCP solver called `sqlp`. The algorith implemented in `sqlp` is an infeasible primal-dual path-following algorithm, described in detail in Toh *et al.*. The basic idea is that, at each iteration, we first compute a predictor search direction aimed at decreasing the gap between the primal and dual objective values. After that, the algorithm generates a corrector step with the intention of keeping the iterates close to the central path. The most crucial part is to solve a linear system which is especially challenging in situations when a big sparse matrix is included. To increase the speed and effeciency, the sparse matrix package `Matrix` is incorporated to deal with high dimensional large datasets.

The calling syntax of is `sqlp` as follows:

$$\text{output} = \text{sqlp}(\text{blk}, \text{At}, \text{C}, \text{b}, \text{OPTIONS}, \text{X0}, \text{y0}, \text{Z0})$$

Input arguments:

- blk: a list describing the block structure of the SOCP problem, which will be described in detail in the following.
- At, C, b: SOCP data.
- OPTIONS: a list of parameters (optional).
- X0,y0,Z0: an initial iterate.

If the input argument OPTIONS is omitted, default values specified in the function `sqlparameters` are used.
Output arguments:

- x: the optimal solution to the SOCP.
- y,Z: the dual solutions.
- info: summary information.
- runhist: run history.

The names chosen for the output arguments explain their contents. The argument $info$ is a list containing performance information such as $info\$termcode$, $info\$obj$, $info\$gap$, $info\$pinfeas$, $info\$dinfeas$, $info\$cputime$ whose meanings are explained in `sqlp`. The argument runhist is a list which records the history of various performance measures during the run; for example, $runhist\$gap$ records the complementarity gap at each interior-point iteration.

While (X, y, Z) normally gives approximately optimal solutions, if $info\$termcode$ is 1 the problem is suspected to be primal infeasible and (y, Z) is an approximate certificate of infeasibility, with $b^T y = 1$, Z in the appropriate cone, and $A^T y + Z$ small, while if $info\$termcode$ is 2 the problem is suspected to be dual infeasible and X is an approximate certificate of infeasibility, with $\langle C, X\rangle = -1, X$ in the appropriate cone, and AX small.

The implementation requires the user to specify the block structure of the given SOCP problems. Let L be the total number of blocks in the SOCP problem. The block structure of the problem data is described by a list of length L called *blk* . The content of each of the elements of the list is given as follows. If the ith block is a quadratic block consisting of p sub-blocks, of dimensions $q_{i1}, q_{i2}, \cdots, q_{ip}$ such that $\sum_{k=1}^{p} = q_i$, then

$$\begin{aligned} \text{blk\$type[i]} &= \text{“q”} \\ \text{blk\$size[[i]]} &= \text{c}(\text{q}_{\text{i1}}, \text{q}_{\text{i2}}, \cdots, \text{q}_{\text{ip}}) \\ \text{At[[i]]} &= [\text{q}_{\text{i}} \times \text{m}] \\ \text{C[i], X[i], Z[i]} &= [\text{q}_1 \times 1] \end{aligned}$$

If the kth blocks is the linear block, then

$$\begin{aligned} \text{blk\$type[k]} &= \text{“l”} \\ \text{blk\$size[[k]]} &= \text{n}_{\text{l}} \\ \text{At[[i]]} &= [\text{n}_{\text{l}} \times \text{m}] \\ \text{C[i], X[i], Z[i]} &= [\text{n}_{\text{l}} \times 1] \end{aligned}$$

Similarly, if the kth blocks is the unrestricted block, then

$$\begin{aligned} \text{blk\$type[k]} &= \text{“u”} \\ \text{blk\$size[[k]]} &= \text{n}_{\text{u}} \\ \text{At[[i]]} &= [\text{n}_{\text{u}} \times \text{m}] \\ \text{C[i], X[i], Z[i]} &= [\text{n}_{\text{u}} \times 1] \end{aligned}$$

The following example shows how `sqlp` call a data file that is stored in the package

```
> data(sqlpData)
> soln <- sqlp(blk = sqlpData$blk, At = sqlpData$At, C = sqlpData$C,
+     b = sqlpData$b, X0 = sqlpData$X0, y0 = sqlpData$y0, Z0 = sqlpData$Z0)
```

4 The Quadratic Programming solver

The optimization program involved in SVM is called Quadratic Programming (QP) which is a special case of SOCP. Similar to SOCP, the application of QP is not limited to SVM, it

can be used to many other areas as well. The package `dwd` also provides a QP solver based on SOCP for solving the following problem

$$\min -d^T b + \frac{1}{2} b^T D b$$
$$s.t. A^T b \geq b_0.$$

SOCP based QP solver is formatted in a similar way but has been shown to be more efficient than existing QP solver: `solve.QP` in `quadprob` package.

The main routine is `solve_QP_SOCP`, whose calling syntax is as follows:

$$\text{output} = \text{solve\_QP\_SOCP}(\text{Dmat}, \text{dvec}, \text{Amat}, \text{bvec})$$

Input arguments.

- Dmat: matrix appearing in the quadratic function to be minimized.
- dvec: vector appearing in the quadratic function to be minimized.
- Amat: matrix defining the constraints under which we want to minimize the quadratic function.
- bvec: vector holding the values of b_0.

Output arguments.

- solution: vector containing the solution of the QP problem.

The following example shows how `solve_QP_SOCP` call a simulated data file

```
> Dmat <- matrix(0, 3, 3)
> diag(Dmat) <- 1
> dvec <- c(0, 5, 0)
> Amat <- matrix(c(-4, -3, 0, 2, 1, 0, 0, -2, 1), 3, 3)
> bvec <- c(-8, 2, 0)
> solve_QP_SOCP(Dmat, dvec, Amat, bvec)
```

5 References

- Marron, J. S., Todd, M. J. and Ahn, J. (2007) "Distance-Weighted Discrimination", Journal of the American Statistical Association, Vol. 102, No. 480, pp. 1267-1271.
- H. Huang, Y. Liu, Y. Du, C.M. Perou, D.N. Hayes, M.J. Todd, and J.S. Marron, "Multiclass distance weighted discrimination with applications to batch adjustment", submitted.
- K.C. Toh, M.J. Todd, and R.H. Tutuncu, "SDPT3 — a Matlab software package for semidefinite programming", Optimization Methods and Software, 11 (1999), pp. 545–581.
- Alexandros Karatzoglou, Alex Smola, Kurt Hornik, "Kernel-based Machine Learning Lab", CRAN - Package kernlab.
- Berwin A. Turlach, "quadprog: Functions to solve Quadratic Programming Problems", CRAN - Package quadprog