Cross-Validation, Support Vector Machines and Slice Models

Michael C. Ferris and Meta M. Voelker

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD (`ferris,voelker@cs.wisc.edu`)
Permanent address: Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706, USA *

Abstract. We show how to implement the cross-validation technique used in machine learning as a slice model. We describe the formulation in terms of support vector machines and extend the GAMS/DEA interface to allow for efficient solutions of linear, mixed integer and simple quadratic slice models under GAMS.

1 Introduction

Slice models are collections of mathematical programs with the same structure but different data. Mathematically, the k-th program in a slice model is given by

$$\begin{array}{lll} \min_x & f^k(x) & \text{(objective slice)} \\ \text{subject to} & A^k x = b^k & \text{(slice constraints)} \\ & x \in \mathcal{X} & \text{(core constraints)} \end{array} \tag{1}$$

Each problem differs from the next in the collection only in a subset of the constraints, and possibly the objective function. We implicitly assume that much of the difficulty in the problems is hidden in the core constraints; for example, discrete variables, large numbers or difficult constraints may all appear as core constraints.

In previous work [3], we described the GAMS/DEA interface that was originally developed to solve large scale Data Envelopment Analysis (DEA) models [2]. In fact, the GAMS/DEA interface provides a way to define and solve more general slice models in GAMS efficiently. Normally, a slice model would be defined and solved in a loop over k, requiring the model to be generated multiple times. However, with the GAMS/DEA interface, the slice constraints are identified by a special alias (`slice`) and are generated all at once. The GAMS/DEA interface then defines the individual problems and passes them to the underlying solver CPLEX [6]. In this way, individual problems are not re-generated, but are instead defined as data modifications of each other, reducing overall model generation time. Furthermore, solutions

* This material is based on research partially supported by the National Science Foundation Grant CCR-9972372, the Air Force Office of Scientific Research Grant F49620-01-1-0040, Microsoft Corporation and the Guggenheim Foundation

can be used as starting points in subsequent solves to speed up overall execution time.

This paper extends the interface to allow efficient testing of support vector machine problems using cross-validation. We have made two enhancements to the interface that allow this technique to be implemented efficiently within a modeling language. Firstly, we provide some simple tools and examples that enable the collection of models arising in cross-validation to be efficiently communicated to the solver. Secondly, we allow quadratic programs (and hence standard support vector machines) to be explicitly formulated as slice models, in addition to linear and mixed integer programs.

2 Example: Breast Cancer Diagnosis

We focus on an example involving the classification of patients' breast tumors as either malignant or benign. We make use of a support vector machine, trained on a set of data for which the outcome is already known. For the data, we use the Wisconsin Diagnosis Breast Cancer Database (WDBC) [7], consisting of 569 patients and 30 features. In order to test the classifier, the model makes use of cross-validation, leaving a portion of the data out of the training for use as testing data.

We solve the standard linear support vector machine model from [1]:

$$\begin{array}{ll} \min_{w,\gamma,z} & \frac{1}{2}\|w\|_2^2 + Ce^T z \\ \text{subject to} & D(Aw-\gamma)+z \geq 1, \quad z \geq 0 \end{array} \tag{2}$$

Here, A is a matrix containing the training data (patients by features) and D is a diagonal matrix with values ± 1, denoting benign tumors $(+1)$ and malignant tumors (-1) for the corresponding patients. C is a parameter weighting the importance of maximizing the margin between the classes versus minimizing the misclassification error (z). The solution w and γ are used to define a separating hyperplane $\{x|w^T x = \gamma\}$ to distinguish between (unseen) benign and malignant data points.

Prior to solving the quadratic programming (QP) model (2), we first consider a modified version that can be converted into a linear program:

$$\begin{array}{ll} \min_{w,\gamma,z} & \|w\|_1 + Ce^T z \\ \text{subject to} & D(Aw-\gamma)+z \geq 1, \quad z \geq 0 \end{array} \tag{3}$$

Model (3) replaces the Euclidean-norm margin measurement of (2) with the sup-norm measurement (resulting in the minimization of the 1-norm) and can be converted into a linear program by adding additional variables, y:

$$\begin{array}{ll} \min_{w,\gamma,z,y} & e^T y + Ce^T z \\ \text{subject to} & D(Aw-\gamma)+z \geq 1, \quad z \geq 0 \\ & y \geq w \geq -y \end{array} \tag{4}$$

3 GAMS Formulations for the Linear Model

Models (2) and (4) are not slice models per se. They become slice models under cross-validation, where they are solved multiple times on different pieces of data. In these cases, only the data A and D vary between solves, appropriately fitting the definition of slice models.

Figure 1 shows the original GAMS formulation for 10-fold cross-validation applied to (4). Only training is carried out in this model, displaying values that define the separating hyperplane (w, γ) for later testing. The equations are defined over the training sets, generated dynamically from the testing sets inside the solve loop. The testing sets are defined by calling the batch include file `gentestset.inc` (figure 2). A and D are defined in `wdbc.gms`.

Figure 3 shows the GAMS/DEA formulation for the same model (where identical lines above the problem definition have been omitted). The major differences between the original and GAMS/DEA formulations are the lack of the solve loop and training sets in the latter. Also, the GAMS/DEA solver is used with an options file `dea.opt` (figure 4).

The solve loop is eliminated in the GAMS/DEA formulation by the way in which the interface handles the data. All of the data is passed to the interface initially and the key name `slice` is used to determine which equations/data belong to which problems. `slice` is aliased to the set that defines the individual problems (in this case, the fold set). Equations that change between problems (such as `sep_def`) must be declared over `slice`.

The training sets are eliminated from the formulation since they are defined implicitly by deleting each testing set from the collection of testing sets in turn. This is an example of a deletion-style slice model (denoted in the options file by `slicetype 0`). DEA-style slice models are addition models: models in which data is added to a core set in order to define a specific problem; addition models are the default slice type in the interface. However, cross-validation models are more appropriately defined by *deleting* slices from the entire (core) set of data.

In the GAMS/DEA formulation, the testing sets are defined prior to the solve statement (since all of the data is passed to the interface at once), and are indexed by `p`. In addition, the equations `sep_def` that depend upon the testing sets are indexed by `p`. By creating a mapping from the slice/fold set `p` to a subset of the patient set `i`, the corresponding (testing) equations are deleted from each of the problems in turn.

Note that equation `obj_def` changes from a sum over the training set to a sum over the entire set in the GAMS/DEA formulation. Variables related to the testing sets in `obj_def` will be eliminated (since they do not appear elsewhere in the model). By default, the GAMS/DEA interface ensures that the number of equations in each problem is the same; most slice models have this property. However, in cross-validation, `card(p)` may not evenly divide `card(i)`. The last problem may have fewer equations than the rest, so we use the option `eqnchk` to turn off the automatic equation check.

```
$title Ten-fold cross-validation example
$eolcom !

$setglobal num_folds 10
set p /1*%num_folds%/;  ! folds to perform
! set i /1*569/;        ! patients (declared in data file)
! set k /1*30/;         ! features (declared in data file)

! Read in data
$include "wdbc.gms"
parameter C /1/;

! Declare testing and training sets
set test(i);
set trai(i);

! Define problem
positive variables z(i), y(k);
variables obj, w(k), gamma;

equations obj_def, sep_def(i), bd1(k), bd2(k);
obj_def..  obj =e= C*sum(trai, z(trai)) + sum(k,y(k));
sep_def(trai)..
           D(trai)*(sum(k, A(trai,k)*w(k)) - gamma) + z(trai) =g= 1;
bd1(k)..   y(k) =g= -w(k);
bd2(k)..   y(k) =g= w(k);

model train /all/;

! Solve
loop(p,

! Generate testing and training sets
$batinclude gentestset.inc i
trai(i) = not test(i);

solve train using lp minimizing obj;
display w.l,gamma.l;

);
```

Fig. 1. The original GAMS formulation for cross-validation applied to model (4).

```
$set i_num floor(card(i) / %num_folds%)

test(%1) = no;

test(%1)$((ord(i) > %i_num%*(ord(p)-1)) and
            (ord(i) <= %i_num%*ord(p))) = yes;
! put leftovers in last set
test(%1)$((ord(p) eq %num_folds%) and
            (ord(i) > %i_num%*%num_folds%)) = yes;
```

Fig. 2. The batch include file `gentestset.inc` for generating the testing sets.

```
! Define problem
alias(p,slice);
positive variables z(i), y(k);
variables obj, w(k), gamma;

equations obj_def, sep_def(i,slice), bd1(k), bd2(k);
obj_def..  obj =e= C*sum(i, z(i)) + sum(k,y(k));
sep_def(i,p)$test(i,p)..
           D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) =g= 1;
bd1(k)..   y(k) =g= -w(k);
bd2(k)..   y(k) =g= w(k);

model train /all/;

! Solve
option lp = dea;  train.optfile = 1;

! Generate testing sets (to be deleted in each problem)
loop(p,

$batinclude gentestset.inc "i,p"

);

solve train using lp minimizing obj;
```

Fig. 3. The GAMS/DEA formulation for cross-validation applied to model (4).

```
slicetype 0
eqnchk 0
primval w,gamma
```

Fig. 4. The options file for the GAMS/DEA formulation.

4 Cross-Validation Testing

Since only one call is made to the solver, the resulting listing file will contain only one set of solutions (for the last problem). Solutions for all of the problems are written to the file `dea.sol`. Inside `dea.sol`, the solutions are given as GAMS parameters, indexed by the slice set. By default, the model status, solver status, and the objective values are written to the solution file.

```
* Ten-fold cross-validation testing file
$eolcom !
$setglobal num_folds 10
set p /1*%num_folds%/;  ! folds to perform

! Read in the data and solution values
$include "wdbc.gms"
$include "dea.sol"

! Declare testing sets
set test(i,p);
loop(p,
$batinclude gentestset.inc "i,p"
);

! Evaluate testing sets
parameter missclass(p);
missclass(p) = sum(i$test(i,p),
  (D(i)*(sum(k,A(i,k)*wval('prim',p,k)) - gammaval('prim',p)) le 0));

display missclass;
```

Fig. 5. GAMS code for calculating the number of misclassified test points.

This means that cross-validation testing cannot be done inside of the training program. Since we need to perform testing in the examples above, we write out `w` and `gamma` through the option `primval`. A separate testing file (figure 5) reads in the solution file `dea.sol` and calculates the number of misclassified test points by counting the number of test points i with $D_{ii}(A_{i\cdot}w - \gamma) \leq 0$. Note that the string "val" is appended to the variable name for the solution value.

5 The GAMS Formulation for the QP Model

We extend GAMS/QPWRAP [5] to solve the QP slice model (2). Under GAMS/QPWRAP, a QP model is formulated as a combination of a linear program (specified in the GAMS model) and a Q matrix (specified in a text file). GAMS/QPWRAP passes the Q matrix to the solver.

```
! Define the Q matrix
parameter q(k,k);  q(k,k) = .5;

! Now write the Q matrix to a file named qmatrix.txt
file qp / qmatrix.txt /;
qp.pc=5; qp.nr=2; qp.nd=13; qp.nw=0; ! Some formatting options

put qp 'Q Matrix for svm';
loop(k, put / 'w' k.tl 'w' k.tl  q(k,k))
putclose;

! Define problem
alias(p,slice);
positive variables z(i);
variables obj, w(k), gamma;

equations obj_def, sep_def(i,slice);
obj_def..  obj =e= C*sum(i, z(i));
sep_def(i,p)$test(i,p)..
           D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) =g= 1;

model train /all/;

! Solve
option lp = deaqp;  train.optfile = 1;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset.inc "i,p"
);

solve train using lp minimizing obj;
```

Fig. 6. The GAMS/DEA formulation for cross-validation applied to model (2).

Figure 6 shows the GAMS formulation for 10-fold cross-validation applied to model (2) (identical lines above the problem definition have been omitted). This formulation is very similar to that of figure 3, with `slice` identifying the slice constraints. It uses the same options file (figure 4). Only the linear portion of model (2) is given explicitly; the quadratic portion is specified by the Q matrix and written to the file `qmatrix.txt` prior to the solver call. In addition, the solver has been changed from `dea` to `deaqp`, that calls GAMS/QPWRAP to import the Q matrix into the model prior to calling GAMS/DEA.

Since the file `qmatrix.txt` must exist prior to the solver call, the ability for GAMS/DEA to solve QP slice models is limited. GAMS/DEA can only solve QP models where the Q matrix does not change between solves, that is, where the Q matrix is part of the core data.

On the WDBC data set, we ran all-but-one testing on both the linear and the quadratic formulations. Since the quadratic model uses a barrier code, previous solutions are not very useful in reducing computational times. For completeness, we cite the following results. Testing of model (4) took 541.71 seconds under GAMS/CPLEX and 269.03 seconds under GAMS/DEA; testing accuracy was 95.08%. Testing of model (2) took 7403.03 seconds under GAMS/CPLEXQP and 7221.74 seconds under GAMS/DEAQP; testing accuracy was 95.42%.

6 The GAMS/DEA Solver

The linear GAMS/DEA solver is specified with the `option lp=dea` statement; the mixed integer GAMS/DEA solver is specified with the `option mip=dea` statement; and the simple QP GAMS/DEA solver is specified with the `option lp=deaqp` statement. Under any of these, GAMS will call the GAMS/DEA interface, that defines and solves the individual problems (based on the generated slice constraints). CPLEX is used as the underlying solver.

GAMS/DEA requires a properly installed GAMS system (distribution 20.1 or newer). In addition, you need a valid GAMS license as well as a GAMS/CPLEX or an ILOG CPLEX 7.1 (or higher) callable library license. It can be obtained from the GAMS Contributed Software web site [4].

References

1. K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah? *SIGKDD Explorations*, 2(2), 2000.
2. W. W. Cooper, L. M. Seiford, and K. Tone. *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*. Kluwer Acaemic Publishers, Boston, 2000.
3. M. C. Ferris and M. M. Voelker. Slice models in general purpose modeling systems. Data Mining Institute Technical Report 00-10, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, December 2000. Available from `http://www.cs.wisc.edu/dmi/tech-reports/`.
4. GAMS Development Corporation. Contributed software. HTML document. `http://www.gams.com/contrib/contrib.htm`.
5. GAMS Development Corporation. Quadratic programs in GAMS. HTML document. `http://www.gams.com/contrib/qpwrap/qpwrap.htm`.
6. ILOG. *CPLEX 7.1 Reference Manual*. Accessed as HTML document, distributed with ILOG CPLEX 7.1 libraries.
7. W. H. Wolberg, W. N. Street, and O. L. Mangasarian. Wisconsin diagnosis breast cancer database (WDBC). FTP documents. Available from `http://www.cs.wisc.edu/~olvi/uwmp/cancer.html`, among other sites.