Paper SAS434-2017

Methods of Multinomial Classification Using Support Vector Machines

Ralph Abbey, Taiping He, and Tao Wang, SAS® Institute Inc.

ABSTRACT

Many practitioners of machine learning are familiar with support vector machines (SVMs) for solving binary classification problems. Two established methods of using SVMs in multinomial classification are the one-versus-all approach and the one-versus-one approach. This paper describes how to use SAS® software to implement these two methods of multinomial classification, with emphasis on both training the model and scoring new data. A variety of data sets are used to illustrate the pros and cons of each method.

INTRODUCTION

The support vector machine (SVM) algorithm is a popular binary classification technique used in the fields of machine learning, data mining, and predictive analytics. Since the introduction of the SVM algorithm in 1995 (Cortes and Vapnik 1995), researchers and practitioners in these fields have shown significant interest in using and improving SVMs.

Support vector machines are supervised learning models that provide a mapping between the feature space and the target labels. The aim of supervised learning is to determine how to classify new or previously unseen data by using labeled training data. Specifically, SVMs are used to solve binary classification problems, in which the target has only one of two possible classes.

The SVM algorithm builds a binary classifier by solving a convex optimization problem during model training. The optimization problem is to find the flat surface (hyperplane) that maximizes the margin between the two classes of the target. SVMs are also known as maximum-margin classifiers, and the training data near the hyperplane are called the support vectors. Thus, the result of training is the support vectors and the weights that are given to them. When new data are to be scored, the support vectors and their weights are used in combination with the new data to assign the new data to one of the two classes.

Many real-world classification problems have more than two target classes. There are several methods in the literature (Hsu and Lin 2002), such as the one-versus-all and one-versus-one methods, that extend the SVM binary classifier to solve multinomial classification problems.

This paper shows how you can use the HPSVM procedure from SAS® Enterprise Miner™ to implement both training and scoring of these multinomial classification extensions to the traditional SVM algorithm. It also demonstrates these implementations on several data sets to illustrate the benefits of these methods.

The paper has three main sections: training, scoring, and experiments, followed by the conclusions. The training section describes how to set up the multinomial SVM training schema. The scoring section discusses how to score new data after you have trained a multinomial SVM. The experiments section illustrates some examples by using real-world data. Finally, the appendices present the SAS macro code that is used to run the experiments.

SUPPORT VECTOR MACHINE TRAINING

Support vector machines (SVMs) are a binary classifier that seeks to find the flat surface (a straight line in two dimensions) that separates the two levels of the target. Figure 1 shows an example of a binary classification problem and the SVM decision surface. In this example, the support vectors consist of the two triangular observations that touch one of the dotted lines and the one hexagonal observation that touches the other dotted line. The dotted lines represent the margin, which indicates the maximum separation between the two classes in the data set.

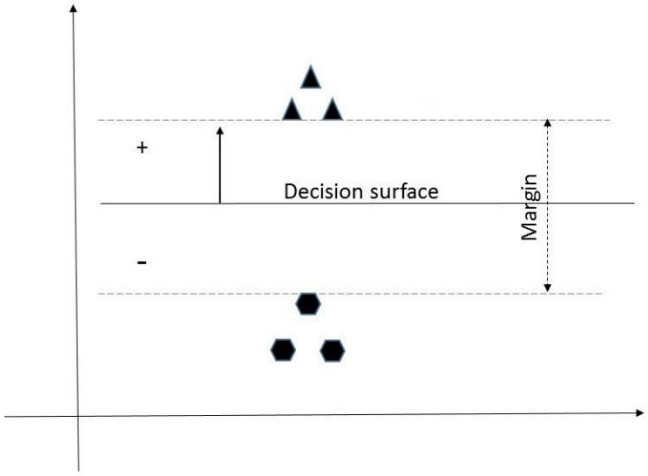

Figure 1. Support Vector Machine Decision Surface and Margin

SVMs also support decision surfaces that are not hyperplanes by using a method called the kernel trick. For the purposes of the examples in this section and the “Support Vector Machine Scoring” section, this paper is limited to referencing only linear SVM models. These sections equally apply to nonlinear SVMs as well. Figure 2 shows an example of two classes that are separated by a nonlinear SVM decision boundary.

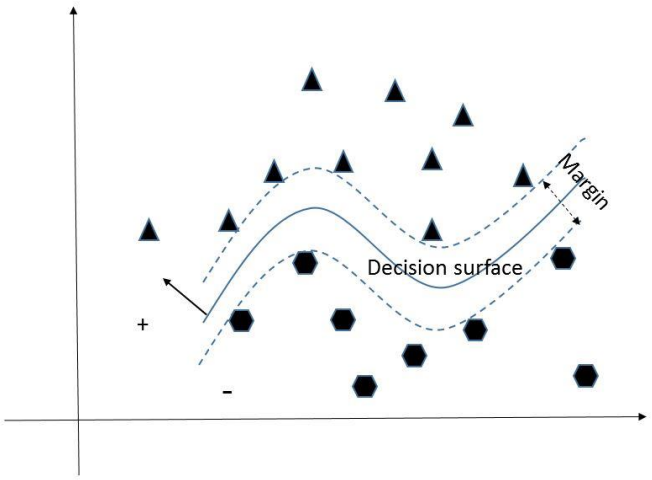

Figure 2. Nonlinear Support Vector Machine Decision Surface and Margin

Multiclass SVMs are used to find the separation when the target has more than two classes. Figure 3 shows an example of a three-class classification problem. Here the classes are triangle, diamond, and hexagon.

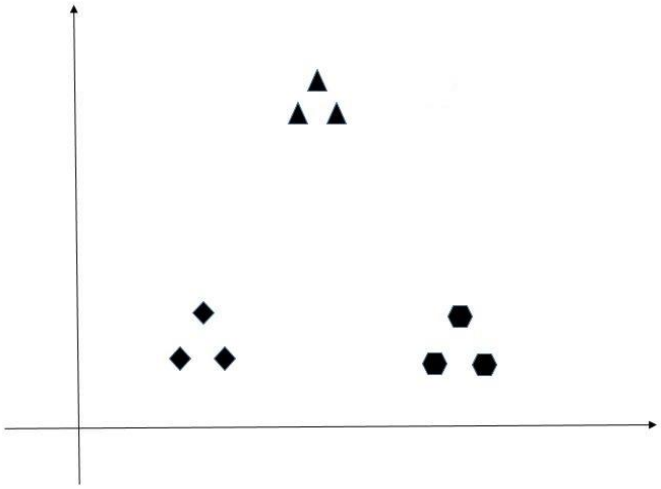

Figure 3. Example of Three Classes: Triangles, Diamonds, and Hexagons

When the data contain more than two classes, a single flat surface cannot separate each group from the others. However, several surfaces can partition the observations from each other. How you find the surfaces depends on your approach in the multiclass SVM: one-versus-all or one-versus-one.

When you are using the HPSVM procedure to solve multinomial classification problems, you first need to create a dummy variable for each class of the target variable. The dummy variable for a particular class is defined to be either 0 when an observation is not of that class or 1 when an observation is of that class. Code to create the dummy variables is presented in the SAS_SVM_ONE_VS_ALL_TRAIN and SAS_SVM_ONE_VS_ONE_TRAIN macros in Appendix B. An example that uses the data in Figure 3 might look something like this:

```
data ModifiedInput;
set Input;
   if (class = "Triangle") then do;
      class_triangle = 1;
   end;
   else do;
      class_triangle = 0;
   end;
   if (class = "Diamond") then do;
      class_diamond = 1;
   end;
   else do;
      class_diamond = 0;
   end;
   if (class = "Hexagon") then do;
      class_hexagon = 1;
   end;
   else do;
      class_hexagon = 0;
   end;
run;
```

When the input data have the dummy variables, the data are ready for you to train using the one-versus-all or one-versus-one method.

One-versus-All Training

The one-versus-all approach to multiclass SVMs is to train *k* unique SVMs, where you have *k* classes of the target. Figure 4, Figure 5, and Figure 6 show the three one-versus-all scenarios for training a multiclass SVM on the example from Figure 3.

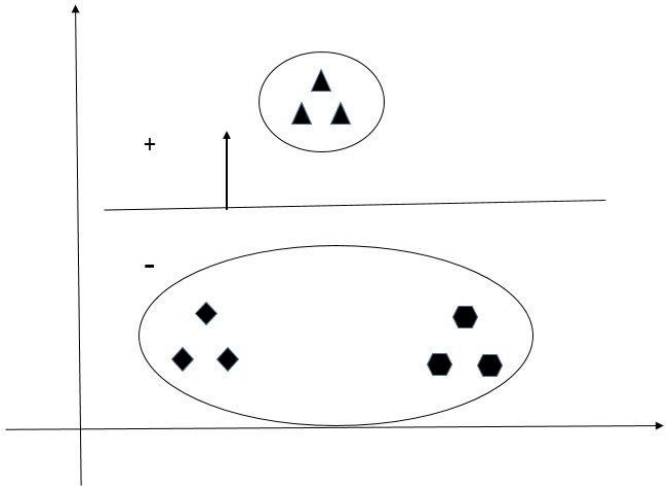

Figure 4. One-versus-All Training: Triangles versus All

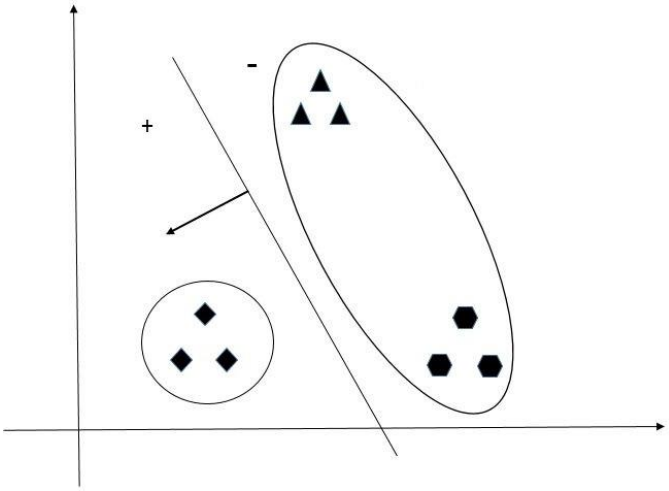

Figure 5. One-versus-All Training: Diamonds versus All

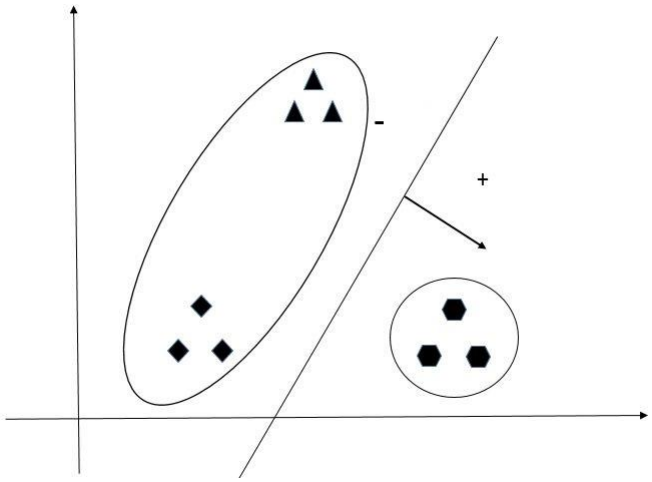

Figure 6. One-versus-All Training: Hexagons versus All

To set up the one-versus-all training by using the HPSVM procedure, you first need to add dummy variables to the input data set, as described previously. The dummy variable that corresponds to Figure 4 has a 1 for each triangular observation and a 0 for each diamond-shaped or hexagonal observation. The HPSVM procedure code for Figure 4 might look like this:

```
proc hpsvm data=ModifiedInput(where=(class=triangle OR class=hexagon));
   input <input variables>;
   target class_triangle;
run;
```

You also need to save the procedure score code by using the CODE statement. This enables you to score new observations based on the training that you have already completed. For the one-versus-all method of multinomial SVM training, you need to run the HPSVM procedure *k* times, and each run will have a different dummy variable as the target variable for the SVM. The output that you need for scoring is *k* different DATA step score code files. You can find a discussion of scoring the one-versus-all method in the section “One-versus-All Scoring.”

One-versus-One Training

The one-versus-one approach to multiclass SVMs is to train an SVM for each pair of target classes. When you have *k* classes, the number of SVMs to be trained is $k*(k–1)/2$. Figure 7, Figure 8, and Figure 9 show the three one-versus-one scenarios for training a multiclass SVM on the example from Figure 3.

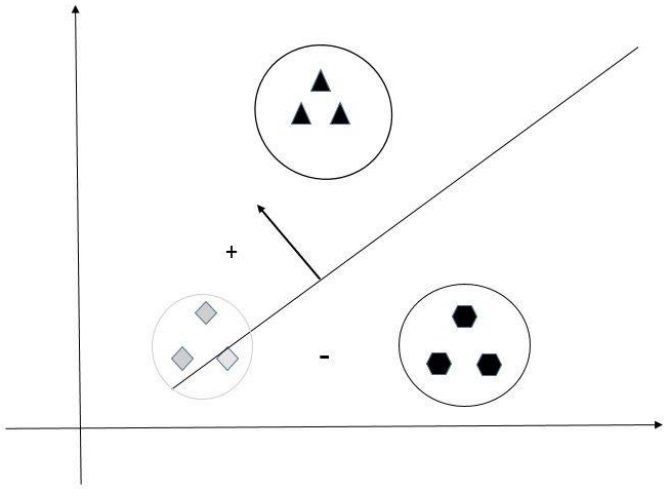

Figure 7. One-versus-One Training: Triangles versus Hexagons

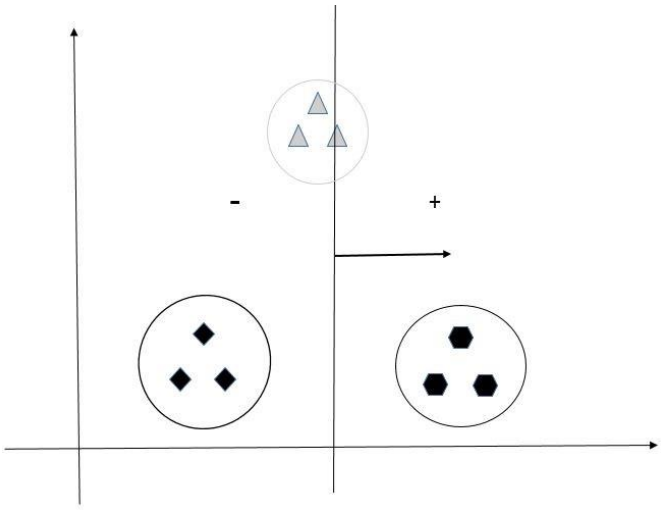

Figure 8. One-versus-One Training: Hexagons versus Diamonds

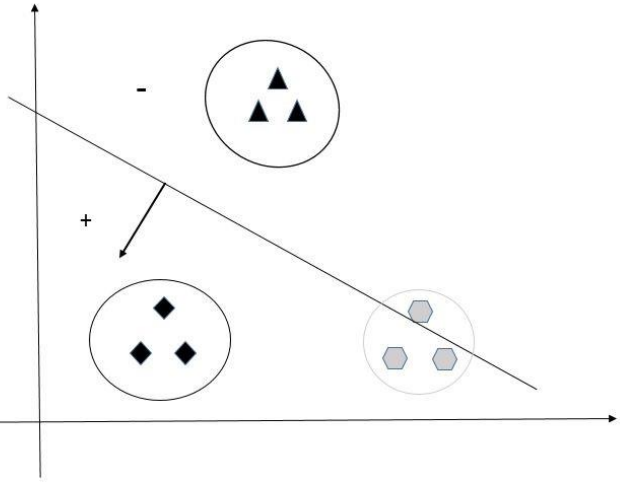

Figure 9. One-versus-One Training: Diamonds versus Triangles

As these three figures show, when you are using the one-versus-one training method of multinomial classification, you ignore any of the data that are not in the current comparison. For this example, you have three comparisons: triangular versus hexagonal observations, hexagonal versus diamond-shaped observations, and diamond-shaped versus triangular observations. In each of these cases, the third class is ignored when you create the SVM model.

To perform this method by using the HPSVM procedure, you first need to create the dummy variables as previously indicated. To ensure that you compare only the proper observations, you also need to subset the input data by using the WHERE= option. An example of the code for triangular versus hexagonal observations might look like this:

```
proc hpsvm data=ModifiedInput(where=(class=triangle OR class=hexagon);
   input <input variables>;
   target class_triangle;
run;
```

As in the one-versus-all method, you need to save the procedure score code by using the CODE statement. This enables you to score new observations based on the training that you have already completed. For the one-versus-one method of multinomial SVM training, you need to run PROC HPSVM $k*(k–1)/2$ times. Each run consists of a different pair of target classes that are compared. The output that you need for scoring is $k*(k–1)/2$ different DATA step score code files. There are two ways to score the one-versus-one training; they are detailed in the sections “One-versus-One Scoring” and “Directed Acyclic Graph Scoring.”

SUPPORT VECTOR MACHINE SCORING

Scoring by using SVMs is the process of using a trained model to assign a class label to a new observation. In the case of the HPSVM procedure, the DATA step score code contains the information from the SVM model and enables you to score new observations. A new example observation, star, has been added to the previous example to illustrate scoring. This is shown in Figure 10.

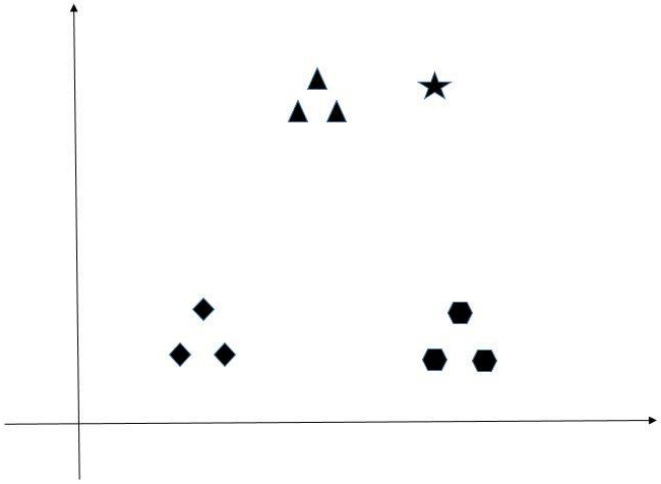

Figure 10. Example Data, with a New Observation (Star) to Be Scored

One-versus-All Scoring

The output from the one-versus-all scoring is *k* DATA step score code files, one for each class of the multinomial target. When you are determining the class of a new data observation, you need to score the observation by using each saved score code.

To assign a class label to a new observation, you need to score the observation scored according to each SVM model. In this way, the new observation will have an assigned probability for each class of the

target. If the observation is on the negative side of the dividing hyperplane, then its probability is less than 0.5. If it is on the positive side of the dividing hyperplane, then its probability is greater than 0.5.

In this example, the hyperplanes shown in Figure 4, Figure 5, and Figure 6 illustrate that the star point will have the highest probability of assignment to the triangular class.

When you are using the PROC HPSVM score code for each class of the target, new data are assigned a probability that the observation is of that target class. To determine which target class is the correct label, you choose the one that has the highest probability. SAS macro code to do this is presented in the SAS_SVM_ONE_VS_ALL_SCORING macro in Appendix B.

One-versus-One Scoring

The output from one-versus-one scoring is $k*(k-1)/2$ DATA step score code files, one for each pairwise comparison of target classes of the multinomial target. When you are determining the class label of a new data observation, you need to score the observation by using each saved score code.

In one-versus-all scoring, each SVM model answers the question, Does this belong to the class or not? In one-versus-one scoring, each SVM model answers a different question: Is this more of class A or class B? Thus, using the maximum probability, as in one-versus-all scoring, is not the appropriate way to determine the class label assignment.

In one-versus-one scoring, a common method of determining this assignment is by voting. Each observation is assigned a class label for each SVM model that is produced. The label that the observation is assigned the most is considered the true label.

When you are using PROC HPSVM, use the score code to score the new data for each one-versus-one SVM model. Then, for each class of the multinomial target, check to see whether that class has the most votes. If it does, then assign that class as the label for the target. When you have a tie, you can assign the class randomly, or as shown in this paper, you can assign the class by using the first class in the sorted order. SAS macro code to perform one-versus-one scoring is presented in the SAS_SVM_ONE_VS_ONE_SCORING macro in Appendix B.

Directed Acyclic Graph Scoring

The directed acyclic graph (DAG), which was first presented in Platt, Cristianini, and Shawe-Taylor (2000), is a scoring approach that uses the same training as the one-versus-one scoring method. In this case, each observation is scored only $k-1$ times, even though $k*(k-1)/2$ SVM models are trained. The training scheme for a four-class example is shown in Figure 11. In this illustration, a new observation starts at the top of the graph and is scored using the 1 vs. 4 SVM model. Then, depending on the outcome, the observation traverses the graph until it reaches one of the four class labels.

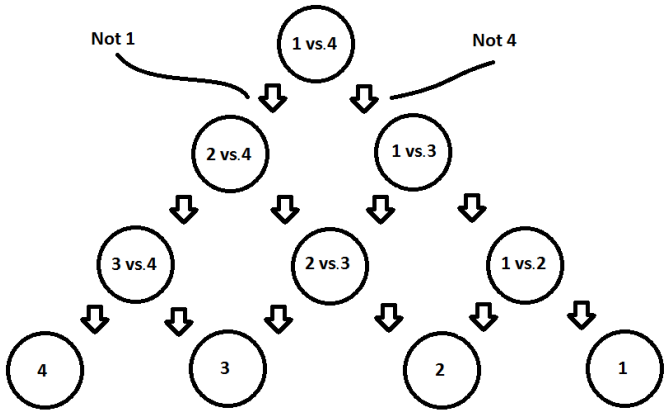

Figure 11. Directed Acyclic Graph Scoring Flow

The DAG method first runs the scoring from the one-versus-one SVM model that compared the first and last classes of the target (the order should be fixed, but the order does not matter). If the SVM model

assigns the observation to the first class, then the last class can be ruled out. Thus the DAG method seeks to recursively exclude possible target class labels until only one label is left. That label becomes the class label for the new observation.

Each one-versus-one model is represented in the DAG. However, the number of times the observation is scored is only k–1, because as it is scored, it flows down the graph.

When you are using PROC HPSVM score code to run the DAG method, you need all the score code files from the one-versus-one training. To score recursively, you need to create two output data sets from each input set, in which you assign each observation to one of the two output data sets based on the predicted target class from the SVM model. SAS macro code to perform DAG scoring is presented in the SAS_SVM_DAG_SCORING macro in Appendix B.

EXPERIMENTS

This section presents a few brief examples that were run using the setup code in Appendix A and the macro code in Appendix B. All the runs of the HPSVM procedure use the procedure defaults, except that the kernel is chosen to be polynomial with degree 2 instead of linear, which is the default.

Table 1 lists the data sets that are used in the experiments. Many of these data sets are available in SAS Enterprise Miner. The **Wine** data set is from the UCI Machine Learning Repository (Lichman 2013).

Simulated data were created to run experiments with larger numbers of observations, input variables, and target classes. The target variable in the simulated data has approximately equal class sizes among the seven classes. In addition, only 9 of the 27 variables are correlated with the target levels, but these correlated variable also have large amounts of randomness.

The HPSVM procedure supports multithreading on a single machine as well as distributed computation. These experiments were run using a single desktop machine. Absolute times vary with hardware and setup, but the relative times provide important insight into how the different methods of multinomial classification compare with each other.

Data Set	Number of Observations	Number of Input Variables	Target Variable	Number of Target Classes	Location
Iris	150	4	Species	3	SASHELP.IRIS
Wine	178	13	Cultivar	3	UCI ML Repository
Cars	428	12	Type	6	SASHELP.CARS
German Credit	1000	20	employed	5	SAMPSIO.DMAGECR
Simulated 10K	10000	27	t	7	Simulated data

Table 1. Data Sets Used in the Experiments, along with Table Metadata

Table 2 shows the training and scoring times for each method on each data set. One-versus-one training is used for both one-versus-one scoring and DAG scoring. When the number of target classes is larger, such as in the **Cars** data set or the simulated data, the one-versus-one training requires more time to complete than the one-versus-all training. This is because there are $k*(k–1)/2$ models that require training for the one-versus-one method, compared to only k models for the one-versus-all method. The number of models that are trained is slightly offset by the fact that each of the models trained in the one-versus-one method uses fewer data than the models trained in the one-versus-all method, but as the number of target classes increases, the one-versus-one method takes more time.

Data Set	Training (sec)		Scoring (sec)		
	One-versus-All Method	One-versus-One Method	One-versus-All Method	One-versus-One Method	DAG Method
Iris	*1*	*1*	*1*	*1*	*1*
Wine	*1*	*1*	*< 1*	1	*< 1*
Cars	*3*	4	*1*	3	2
German Credit	11	*7*	*1*	3	2
Simulated 10K	*67*	76	*2*	7	5

Table 2. Timing for Running the Two Training and Three Scoring Methods on the Data Sets

Table 3 shows the misclassification rate for each data set and each scoring method. For each data set, the one-versus-one method has the best classification rate, followed very closely by the DAG method's classification rate. The one-versus-all method's classification rate is lower than that of the one-versus-one and DAG methods, especially on the larger data sets.

Data Set	One-versus-All Classification Rate (%)	One-versus-One Classification Rate (%)	DAG Classification Rate (%)
Iris	96.00	*96.67*	*96.67*
Wine	*100*	*100*	*100*
Cars	84.81	*87.38*	*87.38*
German Credit	72.5	*76.6*	76.3
Simulated 10K	70.07	*78.06*	78.04

Table 3. Classification Rate for Running the Three Different Scoring Methods on the Data Sets

CONCLUSION

This paper explains how to extend the HPSVM procedure for scoring multinomial targets. Two approaches to extending the SVM training are the one-versus-all and one-versus-one methods. When you are scoring the SVM model, you also have the option to use directed acyclic graphs (DAGs) to score the one-versus-one trained models.

The paper applies one-versus-all and one-versus-one training to several data sets to illustrate the strengths and weaknesses of the methods. The one-versus-one method does better at classifying observations than the one-versus-all method. This benefit is balanced by the fact that as the number of target classes increases, one-versus-one training takes longer than one-versus-all training. The scoring times are also longer for the one-versus-one and DAG methods than for the one-versus-all method. The DAG method runs faster than one-versus-one scoring, with only a marginal decrease in accuracy.

The paper also presents SAS macro code to perform the various multinomial classifications.

APPENDIX A

Before running the SAS macro code in Appendix B, you need to run the setup information. The following example does this for the **Iris** data set:

```
*** the training macros create several SAS data sets and some files;
*** to ensure that nothing is overwritten, create a new directory;
***     or point to an existing empty directory;
*** set the output directory below;
%let OutputDir = U:\SGF2017\; *change as needed;
x cd "&OutputDir";
libname l "&OutputDir";

*** set the target variable;
*** also set the input and score data sets;
*** you can change the score data every time you want to score new data;
%let Target    = Species; *case-sensitive;
%let InputData = sashelp.iris;
%let ScoreData = sashelp.iris;
proc contents data =&InputData out=names (keep = name type length);
run;
data names;
    set names;
    if name = "&Target" then do;
        call symput("TargetLength", length);
        delete;
    end;
run;
*** manually add names to interval or nominal type;
*** id variables are saved from the input data to the scored output data;
%let ID        = PetalLength PetalWidth SepalLength SepalWidth;
%let INPUT_NOM = ;
%let INPUT_INT = PetalLength PetalWidth SepalLength SepalWidth;
%let ID_NUM        = 4;
%let INPUT_NOM_NUM = 0;
%let INPUT_INT_NUM = 4;
*** PROC HPSVM options for the user (optional);
%let Maxiter   = 25;
%let Tolerance = 0.000001;
%let C         = 1;
```

APPENDIX B

The following macros include the one-versus-all training, one-versus-one training, one-versus-all scoring, one-versus-one scoring, and DAG scoring macros. The dummy variable creation is included in the one-versus-all and one-versus-one training macros and has been commented.

```
%macro SAS_SVM_ONE_VS_ALL_TRAIN();
*** separate the target for information-gathering purposes;
data l.TargetOnly;
    set &InputData;
    keep &Target;
    if MISSING(&Target) then delete;
run;
proc contents data = l.TargetOnly out=l.TType(keep = type);
run;
data _NULL_;
    set l.TType;
```

```
    call symput("TargetType", type);
run;
*** get the number of levels of the target;
proc freq data=l.TargetOnly nlevels;
    ods output nlevels=l.TargetNLevels OneWayFreqs=l.TargetLevels;
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** create a column for each level of the target;
*** the value of the column is 1 if the target is that level, 0 otherwise;
data l.ModifiedInput;
    set &InputData;
    _MY_ID_ = _N_;
    %do i=1 %to &n;
        %if (&TargetType = 1) %then %do;
            if MISSING(&Target) then do;
                  &Target.&&level&i = .;
              end;
            else if (&Target = &&level&i) then do;
                &Target.&&level&i = 1;
            end;
            else do;
                &Target.&&level&i = 0;
            end;
        %end;
        %else %if (&TargetType = 2) %then %do;
            if MISSING(&Target) then do;
                  &Target.&&level&i = .;
              end;
            else if (&Target = "&&level&i") then do;
                &Target.&&level&i = 1;
            end;
            else do;
                &Target.&&level&i = 0;
            end;
        %end;
    %end;
run;
*** run an svm for each target. also save the scoring code for each svm;
%do i=1 %to &n;
    %let Target&i = &Target.&&level&i;

    data _NULL_;
        length svmcode $2000;
        svmcode  = "&OutputDir"!!"svmcode"!!"&i"!!".sas";
        call symput("svmcode"||left(trim(&i)), trim(svmcode));
    run;
```

```
    proc hpsvm data = l.ModifiedInput tolerance = &Tolerance c = &C
maxiter = &Maxiter nomiss;
        target &&Target&i;
        %if &INPUT_INT_NUM > 0 %then %do;
            input &INPUT_INT / level = interval;
        %end;
        %if &INPUT_NOM_NUM > 0 %then %do;
            input &INPUT_NOM / level = nominal;
        %end;
        *kernel linear;
        kernel polynomial / degree = 2;
        id _MY_ID_ &Target;
        code file = "&&svmcode&i";
    run;
%end;
*** this table lists all the svm scoring files;
data l.CodeInfoTable;
    length code $2000;
    %do i=1 %to &n;
        code = "&&svmcode&i";
        output;
    %end;
run;
%mend SAS_SVM_ONE_VS_ALL_TRAIN;


%macro SAS_SVM_ONE_VS_ONE_TRAIN();
*** separate the target for information-gathering purposes;
data l.TargetOnly;
    set &InputData;
    keep &Target;
    if MISSING(&Target) then delete;
run;
proc contents data = l.TargetOnly out=l.TType(keep = type);
run;
data _NULL_;
    set l.TType;
    call symput("TargetType", type);
run;
*** get the number of levels of the target;
proc freq data=l.TargetOnly nlevels;
    ods output nlevels=l.TargetNLevels OneWayFreqs=l.TargetLevels;
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** create a column for each level of the target;
*** the value of the column is 1 if the target is that level, 0 otherwise;
data l.ModifiedInput;
```

```
    set &InputData;
    _MY_ID_ = _N_;
    %do i=1 %to &n;
        %if (&TargetType = 1) %then %do;
            if MISSING(&Target) then do;
                  &Target.&&level&i = .;
              end;
            else if (&Target = &&level&i) then do;
                &Target.&&level&i = 1;
            end;
            else do;
                &Target.&&level&i = 0;
            end;
        %end;
        %else %if (&TargetType = 2) %then %do;
            if MISSING(&Target) then do;
                  &Target.&&level&i = .;
              end;
            else if (&Target = "&&level&i") then do;
                &Target.&&level&i = 1;
            end;
            else do;
                &Target.&&level&i = 0;
            end;
        %end;
    %end;
run;
*** run an svm for each target. also save the scoring code for each svm;
%do i=1 %to &n;
    %do j=%eval(&i+1) %to &n;
        %let Target&i = &Target.&&level&i;
        %let Target&j = &Target.&&level&j;

        data _NULL_;
            length svmcode $2000;
            svmcode  = "&OutputDir"!!"svmcode"!!"&i"!!"_"!!"&j"!!".sas";
            call symput("svmcode&i._"||trim(left(&j)), trim(svmcode));
        run;

        proc hpsvm data = l.ModifiedInput(where=(&&Target&i=1 OR
&&Target&j=1)) tolerance = &Tolerance c = &C maxiter = &Maxiter nomiss;
            target &&Target&i;
            %if &INPUT_INT_NUM > 0 %then %do;
                input &INPUT_INT / level = interval;
            %end;
            %if &INPUT_NOM_NUM > 0 %then %do;
                input &INPUT_NOM / level = nominal;
            %end;
            *kernel linear;
            kernel polynomial / degree = 2;
            id _MY_ID_ &Target;
            code file = "&&svmcode&i._&j";
        run;
    %end;
%end;
*** this table lists all the svm scoring files;
data l.CodeInfoTable;
```

```
    length code $2000;
    %do i=1 %to &n;
        %do j=%eval(&i+1) %to &n;
            code = "&&svmcode&i._&j";
            output;
        %end;
    %end;
run;
%mend SAS_SVM_ONE_VS_ONE_TRAIN;


%macro SAS_SVM_ONE_VS_ALL_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
    set l.TType;
    call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
    set l.CodeInfoTable;
    i = _N_;
    call symput("svmcode"||left(trim(i)), trim(left(right(code))));
run;
%do i=1 %to &n;
    %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;
%MakeScoredOneVsAll();
%mend SAS_SVM_ONE_VS_ALL_SCORE;

%macro MakeScoredOneVsAll();
data l.ScoredOutput;
    set &ScoreData;
    %if (&TargetType = 2) %then %do;
        length I_&Target $ &TargetLength;
    %end;
    %do i=1 %to &n;
        %inc "&&svmcode&i";
    %end;
    keep
    %do i=1 %to &n;
        P_&&Target&i..1
    %end;
    %if (&ID_NUM > 0) %then %do;
        &ID
```

```
    %end;
    I_&Target &Target;
    _P_ = 0;
    %do i=1 %to &n;
        %if (&TargetType = 1) %then %do;
            if (P_&&Target&i..1 > _P_) then do;
                _P_ = P_&&Target&i..1;
                I_&Target = &&level&i;
            end;
        %end;
        %else %if (&TargetType = 2) %then %do;
            if (P_&&Target&i..1 > _P_) then do;
                _P_ = P_&&Target&i..1;
                I_&Target = "&&level&i";
            end;
        %end;
    %end;
run;
%mend MakeScoredOneVsAll;


%macro SAS_SVM_ONE_VS_ONE_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
    set l.TType;
    call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
    set l.CodeInfoTable;
    i = _N_;
    call symput("svmcode"||left(trim(i)), trim(left(right(code))));
    call symput("numCode", i);
run;
%let k=1;
%do i=1 %to &n;
    %do j=%eval(&i+1) %to &n;
        %let svmcode&i._&j =&&svmcode&k;
        %let k =%eval(&k+1);
    %end;
%end;
%do i=1 %to &n;
    %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;
```

```
%MakeScoredOneVsOne();
%mend SAS_SVM_ONE_VS_ONE_SCORE;

%macro MakeScoredOneVsOne();
data l.ScoredOutput;
    set &ScoreData;
    %do k=1 %to &n;
        V_&&level&k = 0;
    %end;
    %if (&TargetType = 2) %then %do;
        length I_&Target $ &TargetLength;
    %end;
    %else %do;
        length I_&Target 8;
    %end;
run;
%do i=1 %to &n;
    %do j=%eval(&i+1) %to &n;
        data l.ScoredOutput;
            set l.ScoredOutput;
            %inc "&&svmcode&i._&j";
            if (P_&Target&&level&i..1 >= 0.5) then do;
                V_&&level&i = V_&&level&i+1;
            end;
            else do;
                V_&&level&j = V_&&level&j+1;
            end;
            _P_ = 0;
            %if (&TargetType = 1) %then %do;
                %do k=1 %to &n;
                    if (V_&&level&k > _P_) then do;
                        _P_ = V_&&level&k;
                        I_&Target = &&level&k;
                    end;
                %end;
            %end;
            %else %if (&TargetType = 2) %then %do;
                %do k=1 %to &n;
                    if (V_&&level&k > _P_) then do;
                        _P_ = V_&&level&k;
                        I_&Target = "&&level&k";
                    end;
                %end;
            %end;

            drop P_&Target&&level&i..1 P_&Target&&level&i..0
I_&Target&&level&i _P_;
        run;
    %end;
%end;
data l.ScoredOutput;
    set l.ScoredOutput;
    keep
    %do i=1 %to &n;
        V_&&level&i
    %end;
    %if (&ID_NUM > 0) %then %do;
```

```
        &ID
    %end;
    I_&Target &Target;
run;
%mend MakeScoredOneVsOne;


%macro SAS_SVM_DAG_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
    set l.TType;
    call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
    set l.CodeInfoTable;
    i = _N_;
    call symput("svmcode"||left(trim(i)), trim(left(right(code))));
    call symput("numCode", i);
run;
%let k=1;
%do i=1 %to &n;
    %do j=%eval(&i+1) %to &n;
        %let svmcode&i._&j =&&svmcode&k;
        %let k =%eval(&k+1);
    %end;
%end;
%do i=1 %to &n;
    %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;
%MakeScoredDAG();
%mend SAS_SVM_DAG_SCORE;

%macro MakeScoredDAG();
data ScoredOutput1_&n;
    set &ScoreData;
    _temp_IDvar_ensure_not_existing_ = _N_;
run;
%do k=1 %to %eval(&n-1);
    %let i=&k;
    %let j=&n;
    %do m=1 %to &k;
        %let left =%eval(&i+1);
        %let right=%eval(&j-1);
```

```
        data tempL tempR;
            set ScoredOutput&i._&j;
            %inc "&&svmcode&i._&j";
            if (I_&Target&&level&i = 1) then do;
                output tempR;
            end;
            else do;
                output tempL;
            end;
        run;
        %if &m=1 %then %do;
            data ScoredOutput&left._&j;
                set tempL;
            run;
        %end;
        %else %do;
            data ScoredOutput&left._&j;
                set ScoredOutput&left._&j tempL;
            run;
        %end;
        data ScoredOutput&i._&right;
            set tempR;
        run;
        %let i=%eval(&i-1);
        %let j=%eval(&j-1);
    %end;
%end;
data ScoredOutput;
    set
    %do i=1 %to &n;
        ScoredOutput&i._&i.(in = in&i.)
    %end;
    ;
    %if (&TargetType = 2) %then %do;
        length I_&Target $ &TargetLength;
    %end;
    %if (&TargetType = 1) %then %do;
        %do i=1 %to &n;
            if (in&i.) then do;
                I_&Target = &&level&i;
            end;
        %end;
    %end;
    %if (&TargetType = 2) %then %do;
        %do i=1 %to &n;
            if (in&i.) then do;
                I_&Target = "&&level&i";
            end;
        %end;
    %end;
    keep
    %if (&ID_NUM > 0) %then %do;
        &ID
    %end;
    I_&Target &Target _temp_IDvar_ensure_not_existing_;
run;
```

```
proc sort data=ScoredOutput
out=l.ScoredOutput(drop=_temp_IDvar_ensure_not_existing_);
    by _temp_IDvar_ensure_not_existing_;
run;
%do i=1 %to &n;
    %do j=&i %to &n;
        proc delete data=ScoredOutput&i._&j;
        run;
    %end;
%end;
%mend MakeScoredDAG;
```

REFERENCES

Cortes, C., and Vapnik, V. (1995). "Support-Vector Network." *Machine Learning* 20:273–297.

Hsu, C.-W., and Lin, C.-J. (2002). "A Comparison for Multiclass Support Vector Machines." *IEEE Transactions on Neural Networks* 13:415–425.

Lichman, M. (2013). "UCI Machine Learning Repository." School of Information and Computer Sciences, University of California, Irvine. http://archive.ics.uci.edu/ml.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). "Large Margin DAGs for Multiclass Classification." *Advances in Neural Information Processing Systems* 12:547–553.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Ralph Abbey
SAS Institute Inc.
Ralph.Abbey@sas.com

Taiping He
SAS Institute Inc.
Taiping.He@sas.com

Tao Wang
SAS Institute Inc.
T.Wang@sas.com