STAR CLASSIFICATION ANALYSIS

Jeremy Meyer
Department of Statistics
Brigham Young University

1 Introduction

Throughout the universe, there exists an incomprehensible number of stars. Although only a handful are visible to the naked eye, technology advancements have allowed astronomers to see more stars and make accurate measurements about them. Astronomers have grouped stars that share similar properties. They categorized stars into types such as brown, red, and white dwarfs; main sequence stars; and super/hyper giant stars. These categories do not have rigid boundaries, rather they represent clusters of stars seen in the universe. As telescopes and computer vision advance, astronomers gain the ability to quickly collect data on millions of stars. Instead of manually labeling each star, a classifier could automatically label each star based on its observed properties.

In this analysis, we will build a support vector machine (SVM) to classify stars into their appropriate class. Since there are 6 possible classes, we will employ methods that are designed to break a multi-class problem into binary parts. Research for multi-class SVMs is still ongoing, but we will use methods such as one-vs-one (OvO), one-vs-rest or one-vs-all (OvA), and Divide-by 2 (DB2). Using these 3 methods, we will compare the results of all 3 multi-class SVM methods in terms of prediction accuracy. We will also perform a simulation study with similar data comparing the 3 methods. We will explore the effect of group distinctness and class balance on computation time and prediction accuracy.

1.1 Data

The original dataset[1] contains 4 distinct covariates on 240 stars: temperature (K), luminosity (Solar Luminosity), radius (relative to the Sun), and absolute magnitude (M). With these covariates, fitting any multi-class SVM produced perfectly accurate classifiers on testing data. Although there is not anything wrong with this from a practical standpoint, we cannot make comparisons across the 3 methods. Since absolute magnitude and luminosity both measure brightness, we left out absolute magnitude. In some circumstances, the data collection process not ideal and sub-optimal data is still used for analysis. Thus, we will make robust classifiers using a subset of the predictors. There are 6 groups of stars, with 40 in each group. The group averages for the covariates we used are displayed in Table 1. Some distinctions between classes can be seen from the table, like how dwarf stars are much smaller/dimmer than other stars.

Table 1: Averages for the covariates across each star class

Star Class	Temperature	Radius	Luminosity
Brown Dwarf	2997.9	0.110	6.933×10^{-4}
Red Dwarf	3283.8	0.348	5.405×10^{-3}
White Dwarf	13931.4	0.010	2.433×10^{-3}
Main Sequence	16018.0	4.430	3.206×10^{4}
Supergiant	15347.8	51.150	3.018×10^{5}
Hypergiant	11405.7	1366.897	3.092×10^{5}

2 Methods - SVMs

Support vector machines work by creating a hyperplane (dimension P-1) that best separates 2 classes in P dimensions. While we are interested in separating more than 2 classes, the multi-class methods (discussed later) use many binary classifications. The hyperplane can be represented as $\mathcal{H} = \{\boldsymbol{x} : \beta_0 + \boldsymbol{x}^{'}\boldsymbol{\beta} = 0\}$, where $\boldsymbol{x}$ is a point in P space, β_0 is an intercept for the hyperplane, and $\boldsymbol{\beta}$ are the other P hyperplane coefficeints. We say that if the classifier function, $f(\boldsymbol{x_0}) = \beta_0 + \boldsymbol{x_0^{'}}\boldsymbol{\beta}$, is positive, then point $\boldsymbol{x_0}$ is above $\mathcal{H}$. We can also apply this same idea to classes: if $f(\boldsymbol{x_0}) > 0$,

then $\boldsymbol{x_0}$ belongs to a different class than if $f(\boldsymbol{x_0}) < 0$. We will refer to $\beta_0 + \boldsymbol{x_0'}\boldsymbol{\beta}$ as the decision value. The larger the decision value is in magnitude, the further the point is from the hyperplane, or equivalently, the more confident we are in that an object belongs to a certain class.

Let $\boldsymbol{x_i}$ be the vector of covariates for the i^{th} observation and let $y_i \in \{-1, 1\}$ be its corresponding class. The predicted class is $\hat{y_i} = sign(\beta_0 + \boldsymbol{x_i'}\boldsymbol{\beta})$. If all classes are on the correct side, $y_i(f(\boldsymbol{x_i})) > 0 \quad \forall i$. When finding an optimal hyperplane to separate the data, we run into two obstacles:

1. If a hyperplane can separate the data perfectly, there are an infinite number of ways to do so.
2. Typical data is not completely separable and thus a perfectly separating hyperplane does not exist.

To address 1), SVMs finds the hyperplane that maximizes the margin (M), or smallest orthogonal distance of each datapoint to the hyperplane. This is often called the *Maximal Margin Hyperplane*. In cases where the data are perfectly separable, the margin is a measure of space between classes. However, with typical data, it is usually impossible to find such a hyperplane, and even if one did exist, it could potentially lead to overfitting. Because of the constraint of every point being on the correct side, additional data may shift the margin/hyperplane significantly. It may be better to missclassify a few observations for better future prediction.

To address 2), we use a *soft margin classifier* or support vector classifier that separates the points as much as possible. The margin is now loosely defined as an adjacent, parallel region to the hyperplane, and observations can be within the margin or on the wrong side of the hyperplane. Each observation is given a slack variable ϵ_i, which is similar to a residual. $\epsilon_i = 0$ if the observation is on the correct side and outside the margin, $\epsilon_i \geq 1$ if it is on the wrong side, and $\epsilon_i \in (0, 1)$ if it is within the margin on the correct side. Thus, the constraint is now for $y_i(\beta_0 + x_i'\boldsymbol{\beta}) \geq M(1 - \epsilon_i) \quad \forall i$ A support vector is a point that lies either within the margin or on the wrong side. In the e1071 R library, the cost tuning parameter (C), determines how much to penalize the slack variables. Lower values of C allow for more $\epsilon_i > 0$ and more support vectors, and higher values of C allow for less error. As $C \to \infty$, the SVM performs as if the data is completely separable.

Using Lagrangian dualities (where $\hat{\lambda_i}$ is from the lagrangian), it can be shown that the optimal hyperplane is found by

$$\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_i \epsilon_i \quad \text{subject to} \quad y_i(\beta_0 + x_i'\boldsymbol{\beta}) \geq M(1 - \xi_i),\ \epsilon_i \geq 0 \quad \forall i. \tag{1}$$

Which has the hyperplane solution, dependent only on the support vectors,

$$\hat{f}(\boldsymbol{x_0}) = \hat{\beta}_0 + \sum_{i: y_i(\beta_0 + \boldsymbol{x_i}\boldsymbol{\beta}) > M(1-\xi_i)} \hat{\lambda_i} y_i \boldsymbol{x_0'}\boldsymbol{x_i}. \tag{2}$$

Sometimes two classes may not be separable by a line, but by a curve. We can perform a basis function expansion on $\boldsymbol{x_0'}\boldsymbol{x_i}$ from Equation 2 to get non-linear classification boundaries. These expand the covariates to a higher space where it can potentially be more separable. These expansions are called kernels, denoted $K(\cdot, \cdot)$, and some have a tuning parameter γ. Here are some common SVM kernels:

- Linear: $K(\boldsymbol{x_0}, \boldsymbol{x_i}) = \boldsymbol{x_0'}\boldsymbol{x_i}$
- Polynomial (p^{th} order): $K(\boldsymbol{x_0}, \boldsymbol{x_i}) = (1 + \gamma \boldsymbol{x_0'}\boldsymbol{x_i})^p$
- Radial: $K(\boldsymbol{x_0}, \boldsymbol{x_i}) = exp(-\gamma||\boldsymbol{x_0} - \boldsymbol{x_i}||^2)$
- Sigmoid: $K(\boldsymbol{x_0}, \boldsymbol{x_i}) = tanh(\gamma \boldsymbol{x_0'}\boldsymbol{x_i} + 1)$

2.1 Multi-class SVMs

We now describe how to extend binary SVMs to k classes using 3 different methods. All of these methods can generalize to any binary classifier, so we used them for SVMs. One-vs-one (OvO) builds classifiers for all class pairs and then classifies according to the most frequently predicted class. OvO trains $\binom{k}{2}$ classifiers from subsets of the data. For example, if $k = 6$, we build 15 SVMs where only data from classes 1 and 2 are used to build a SVM for the 1,2 class pair.

One-vs-all (OvA) compares each class against all other classes by finding the class with the highest decision value ($\beta_0 + \boldsymbol{x_0'}\boldsymbol{\beta}$). OvA builds k binary classifiers by comparing each class with all of its complimentary classes. For example, if we build a binary SVM classifier by assigning $y_i = 1$ for class 1 and $y_i = -1$ to classes $2, 3, ..., k$, compute the

decision value for a future observation, and find that it's higher than all other $k-1$ decision values from classes $2, 3..., k$, we predict class 1.

Vural and Dy[2] introduce the divide-by-2 method (DB2) that works by sequentially dividing the classes into 2 subsets until each subset is only a single class. The result is similar to a decision tree or dendrogram between all the classes. This method only builds $k-1$ classifiers. For example, let $k = 5$. We may start by classifying between the class subsets $\{1, 2\}$ or $\{3, 4, 5\}$. If we classify the point into the subset $\{1, 2\}$, we then decide between classes 1 and 2. Each prediction is fed through the classifiers until it is predicted to a single class. This method offers flexibility in how to split the classes so that it can be adapted to the context of the problem.

We will use agglomerative clustering methods with class covariate centroids to create the binary decision tree. Agglomerative clustering methods seek to find natural groups based on Euclidean distance. Objects start in their own group and are grouped to other objects closest in distance until every object is in the same group. When objects are grouped together, it is necessary to define the distance between the new cluster and other observations. Ward's linkage defines distances in such a way that maximizes the variance between the clusters relative to the variance within each cluster. We will use ward's linkage because it tends to produce even splits, thus reducing the average number of classifications needed to get to a single class.

Thus, the number of classifiers needed for each of the 3 methods is displayed in Table 2. Although OvO requires many more classifiers, it may actually be faster for small k since it builds each individual SVM using less data. OvA only requires k classifiers, but it has to train on all the data. As k gets large, OvO may be computationally infeasible. DB2 requires the least amount of classifiers and only one classifier needs to train on the full data, though it requires specification on how to split the classes.

Table 2: Number of classifiers needed for k classes

k	3	6	10	20	50	100
OvO	3	15	45	190	1225	4950
OvA	3	6	10	20	50	100
DB2	2	5	9	19	49	99

For numerical stability, the data was centered and scaled before analysis. 25% of the star data was set aside as a testing set and the 75% remaining (or training set) was used to build the SVMs. SVMs require tuning for optimal predictive performance. Since we do not know which kernel, γ, or cost parameter C to use, we used 10-fold cross validation (10-CV) on the training set to find optimal tuning parameters. We performed 10-CV by splitting the training set into 10 random groups, holding out one group, implementing one of the 3 multi-class methods using data combined from all 9 other groups, testing performance by comparing predictions from the held out group with their actual classes, repeating 10 times so that each group has been held out once, and averaging the prediction accuracies across iterations. Using a simple grid search, the optimal we found the optimal tuning parameters for each of the 3 methods by finding the parameters with the highest 10-CV accuracy. Once we have final tuning parameters for all 3 methods, we compared performances using prediction accuracies on the testing set held out at the beginning.

Additionally, we will perform a simulation study to extend our analysis of the star dataset. Specifically, we will look at how the balance and separation of classes affects prediction accuracy and computation time across the 3 methods.

3 Results

We started by first finding the optimal tuning parameters for each kernel. For simplicity, we used the same tuning parameters for all classifiers in the 3 methods. We selected the kernel with the highest cross-validated accuracy from the training set. The results are displayed in Table 3. Note that DB2 and OvO performed relatively well across kernels, whereas OvA struggled to predict well on most kernels. An example graph of finding the optimal tuning parameters is shown in Figure 1. For this data, very high values of C predicted the best. This means that SVMs with lower slack values fit better, and so the data is fairly separable. This can be seen from the high (>90%) CV prediction accuracies.

In order to perform the DB2 method, we must find a way to split the classes in the data. We used the dendrogram in Figure 2. That, we first classify points into the subsets {Brown_Dwarf, Red_Dwarf} or {Hypergiant, Supergiant, White_Dwarf, Main_Sequence}, then for any points classified into the first group, we then use another classifier to assign the points to either a Brown or Red dwarf. The same process follows for the {Hypergiant, Supergiant, White_Dwarf, Main_Sequence} group.

After taking the optimal tuning parameters and kernels from 3, we then compared performances by generating and evaluating class predictions from testing set held out at the beginning. This set was not used to tune or train any SVMs,

so this will give us an estimate on how well these SVMs predict on future data. We found that the DB2 and OvO methods performed the best on the star dataset, but all three predicted with at least 90% accuracy.

Table 3: Cross Validation accuracy (Acc) results. These are the best tuning parameters for each kernel across the 3 methods. For the polynomial kernel, $p = 3$ was used.

(a) One-vs-one CV results

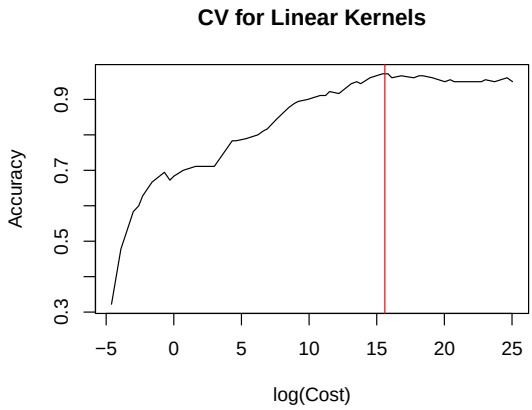
Kernel	C	γ	Acc
Linear	5×10^6	——	97.2%
Polynomial	1×10^7	0.1	95.5%
Radial	500	50	95.5%
Sigmoid	5×10^8	0.01	96.6%

(b) One-vs-All CV results

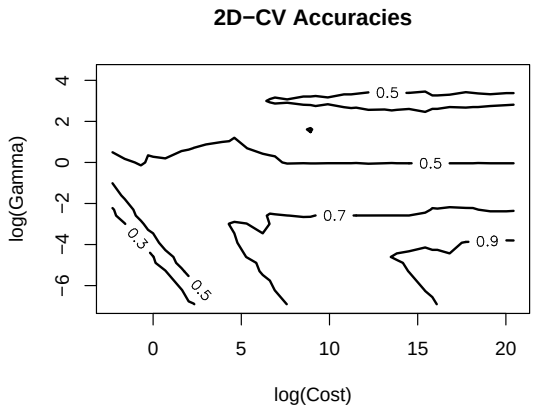
Kernel	C	γ	Acc
Linear	50	——	83.3%
Polynomial	750	2	78.3%
Radial	1×10^7	0.1	93.3%
Sigmoid	5×10^6	0.001	72.2%

(c) DB2 CV results

Kernel	C	γ	Acc
Linear	50000	——	97.8%
Polynomial	5000	.075	95.5%
Radial	5×10^5	0.02	97.2%
Sigmoid	2×10^5	0.075	98.3%



(a) CV for linear kernels



(b) CV for the Sigmoid Kernels

Figure 1: Sample Cross Validation (CV) results for the tuning parameters using the OvO method. Across many methods, high cost values and low gamma values tended to predict best.

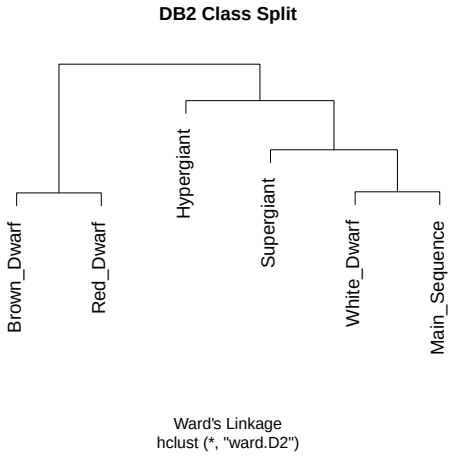


Figure 2: Hierarchical clustering dendrogram using Ward's linkage. Using the DB2 method, the classifiers are used at each split and altogether determine the appropriate class.

Table 4: Final testing set accuracies for the 3 methods

Method	Kernel	C	γ	Accuracy
OvO	Linear	5×10^8	0.01	95.0%
OvA	Raidal	1×10^7	0.1	90.0%
DB2	Sigmoid	2×10^5	0.075	**96.7%**

4 Simulation Study

Now we extend the analysis on the star dataset on simulated data. We simulated 240 datapoints from 6 different classes. The classes will either be balanced (40 in each class), varied (sizes *20*, *40*, *50*, *60*, *40*, *30* respectively), and unbalanced sizes *12*, *20*, *50*, *110*, *30*, *18* respectively). We chose these size numbers so that unbalanced sizes are an exaggeration of the varied sizes.

Table 5: Normal Distribution parameters that generated the covariates.

Parameter	X1	X2	X3	X4
μ	Each group N(0,1)	0,0,-0.2,1,1,3	-.5,0,0,1,2,2	All 0
σ	All 1	1,1,1,2,2,2	Gamma(1,1)	All 1

We generated four covariates, all of which were normally distributed with parameters shown in Table 5. To make fair comparisons, the data were the same for all 3 multi-class methods. X_1 gives each group its own mean, X_2 is similar to the radius variable, X_3 is similar to luminosity, and X_4 is a variable with no regard to class. In order to generate levels of group distinction, we scaled the mean of all covariates by 0.5 (Low), 1 (Moderate), and 2 (High). Thus we will compare the 3 methods across group distinction and class balance using computation time and test set accuracy. Before predicting from the testing set, all of these methods were tuned using a grid search for optimal results. Thus, the computation time includes the 10-CV across a grid of 250 possible values and the time required to predict values from the testing set. We used the class split as defined in Figure 3 for the DB2 method. The results are displayed in Tables 6-11.

Table 6: OvO Test set accuracy

OvO	Balanced	Varied	Unbalanced
Low	33.3%	55.0%	51.7%
Moderate	50.0%	66.7%	71.7%
High	75.0%	80.0%	86.7%

Table 7: Computation time

OvO	Balanced	Varied	Unbalanced
Low	30.1sec	30.2	24.4
Moderate	18.4sec	19.6	12.5
High	12.2sec	12.1	9.5

Table 8: OvA Test set accuracy

OvA	Balanced	Varied	Unbalanced
Low	45.0%	45.0%	58.3%
Moderate	59.7%	58.3%	74.6%
High	63.3%	78.3%	90.0%

Table 9: computation time

OvA	Balanced	Varied	Unbalanced
Low	4:34min	3:39	1:59
Moderate	2:47min	2:03	0:55
High	1:20min	0:59	0:40

Table 10: DB2 Test set accuracy

DB2	Balanced	Varied	Unbalanced
Low	40.0%	45.0%	65.0%
Moderate	60.0%	68.3%	76.7%
High	73.3%	78.3%	78.3%

Table 11: Computation Time

DB2	Balanced	Varied	Unbalanced
Low	2:45min	2:10	1:15
Moderate	1:35min	1:32	1:09
High	0:58min	0:58	0:43

Although the accuracy appears to increase for all methods as the classes get unbalanced, that is a result of the SVMs classifying more datapoints into the largest class. An example can be seen in Table 12. The accuracies in the rarer classes are not as high as the common classes. This could be an indicator of a small sample, so SVMs may need several (more than 12-20 in our simulated dataset) points from all classes for better classification. Overall accuracy was a mixed bag between the 2 methods, but for the simulated data, OvO may be better for more distinct classes, and OvA may be better for data that is hard to separate. For Balanced data that has high separation between classes, DB2 and OvO do

best, similar to what was found with the star dataset. As class separation increases, all methods seem to improve at the same rate.

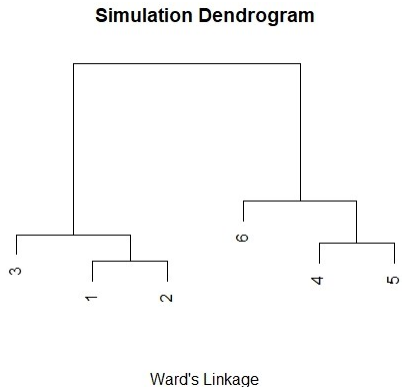


Figure 3: Example dendrogram for DB2 class splits on the simulated data. Most scenarios of class distinction and balance resulted in this dendrogarm.

In terms of computation time, unbalanced datasets also take less time to evaluate. This could be because the SVMs are simply putting most points into a few classes. Computation time was the quickest for OvO. One reason this may be the case is because polished code from the e1071 R library was used for tuning, but it can also be because less data was used for training each classifier. Since the number of classifiers increases exponentially with k for OvO, OvO may take significantly longer for a larger k. DB2 performed twice as fast for data with low separation between classes than OvA, but both DB2 and OvA had roughly the same computation time as class separation increased.

Table 12: Class test set accuracy rates for OvA Moderate spread. The (Size) represents the class distribution in the entire dataset.

Classes	1	2	3	4	5	6
Balanced	20.0%	60.0%	57.1%	88.9%	83.3%	50.0%
(Size)	*40*	*40*	*40*	*40*	*40*	*40*
Varied	33.3%	36.4%	37.5%	92.3%	76.9%	44.4%
(Size)	*20*	*40*	*50*	*60*	*40*	*30*
Unbalanced	40.0%	33.3%	60.0%	96.0%	100.0%	16.7%
(Size)	*12*	*20*	*50*	*110*	*30*	*18*

5 Conclusion

In summary, we found that the DB2 and OvO methods performed the best on the star classification dataset. All 3 multi-class SVM methods were able to separate the groups well, due to the high cost parameters and testing set accuracies over 90%. It appears that these methods are robust after removing a covariate from the data. It would be interesting to see if the SVM is just as predictive with a dataset of thousands of stars.

From the simulation study, we concluded that OvO did best in terms of computation time, followed by DB2. However, this may not hold with a larger number of classes. It may be worth looking into when it is unfeasible to use OvO due to computational intensity. Overall accuracy was a mixed bag between the 3 methods, but OvO performed best on data that had more distinct classes and OvA performed best on data that was harder to separate. Although overall accuracy increased with unbalanced data, all methods suffered to predict well with infrequent classes, but this may just be a reflection of a small number of points in some classes. For future work, we could explore how these methods perform on rare classes as we increase the size of the training set.

References

[1] Star Dataset To Predict Star Types (2019) https://www.kaggle.com/deepu1109/star-dataset

[2] Vural, V. and Dy, J. (2004) A Hierarchical Method for Multi-class Support Vector Machines. In proceedings of The Twenty-First International Conference on Machine Learning (ICML), p. 831-838