Multi-class Support Vector Machines for Star Classification

Jeremy Meyer

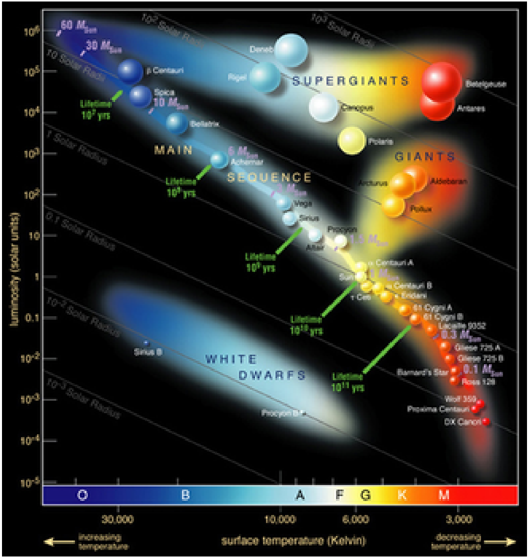
STAT 666 Project

December 12, 2019

Introduction

- SVMs are built for binary classifications.
- We want to extend to problems of multiple classes
- Compare 3 different methods common in literature
- Apply to real-life example (star classication) and simulation.

Star Types


SUPERGIANTS
GIANTS
MAIN
SEQUENCE
WHITE
DWARFS
luminosity (solar units)
surface temperature (Kelvin)
increasing temperature
decreasing temperature
O
B
A
F
G
K
M
30,000
10,000
6,000
3,000
Deneb
Rigel
Betelgeuse
Antares
Canopus
Polaris
Arcturus
Aldebaran
Pollux
Vega
Sirius
Procyon
Altair
Bellatrix
Achernar
Spica
Sirius B
Procyon B
Barnard's Star
Ross 128
Wolf 359
Proxima Centauri
DX Cancri

The Data

Table: Averages for the covariates across each star class

Star Class	N	Temperature (K)	Radius ($R_\odot$)	Luminosity ($L_\odot$)
Brown Dwarf	40	2997.9	0.110	6.933×10^{-4}
Red Dwarf	40	3283.8	0.348	5.405×10^{-3}
White Dwarf	40	13931.4	0.010	2.433×10^{-3}
Main Sequence	40	16018.0	4.430	3.206×10^{4}
Supergiant	40	15347.8	51.150	3.018×10^{5}
Hypergiant	40	11405.7	1366.897	3.092×10^{5}

- Balanced Classes
- $\odot$ = Relative to the Sun

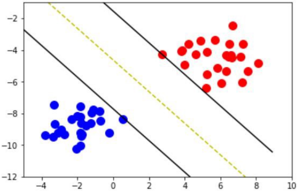
Goals of Analysis

- It would be nice if computers could label thousands of stars for us
- Build a SVM classifier to predict star type
- Compare across 3 different multi-class techniques:
  1. One vs One
  2. One vs All
  3. Divide-by 2

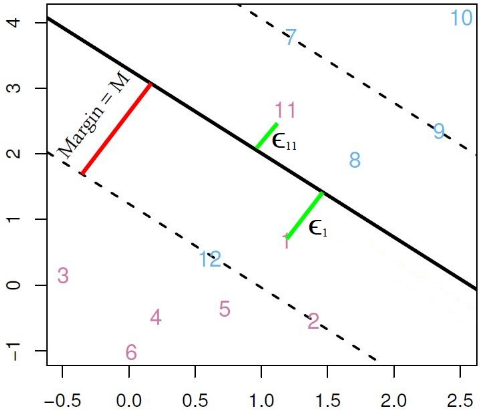
(Brief) SVM review

Main idea: Find the hyperplane that best separates the data into 2 classes

- Hyperplane Equation $\mathcal{H} = \{\boldsymbol{x} : \beta_0 + \boldsymbol{x}'\boldsymbol{\beta} = 0\}$
  - βs $\rightarrow$ estimated hyperplane coefficients
- **Classifier function**: $f(\boldsymbol{x_0}) = \beta_0 + \boldsymbol{x_0}'\boldsymbol{\beta}$
  - If $\beta_0 + \boldsymbol{x_0}'\boldsymbol{\beta} \geq 0 \rightarrow$ Class A
  - If $\beta_0 + \boldsymbol{x_0}'\boldsymbol{\beta} < 0 \rightarrow$ Class B
- **Decision Value**: $\beta_0 + \boldsymbol{x_0}'\boldsymbol{\beta}$
- More positive $\rightarrow$ more like Class A

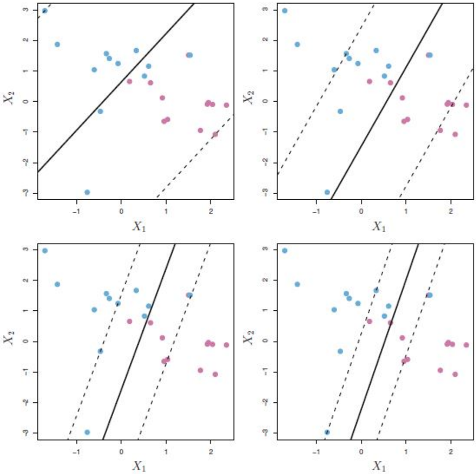


SVM terms



- Each datapoint $\boldsymbol{x_i}$ gets a **slack value** ϵ_i (residual)
- SVM seeks to maximize the margin (M) while minimizing slacks ϵ_i's.
- $\boldsymbol{x_i}$ on wrong side: $\epsilon_i \geq 1$
- $\boldsymbol{x_i}$ is on right side & inside M: $\epsilon \in (0, 1)$
- $\boldsymbol{x_i}$ is on correct side and outside M: $\epsilon_i = 0$

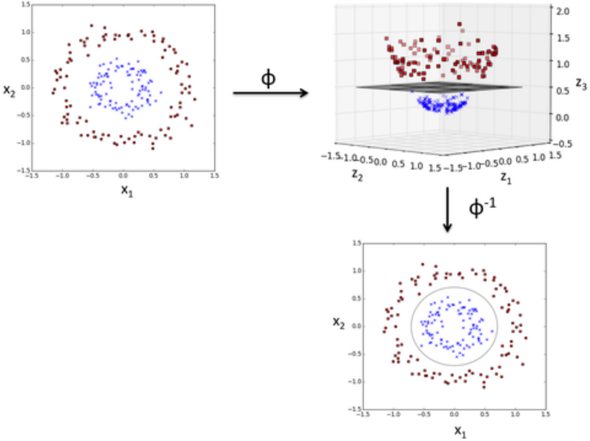
C (Tuning parameter) - e1071 library



Cost (C) determines how much to penalize slack variables
Low C values (top right) high C (bottom left)

Kernels

Sometimes a straight line doesn't cut it...



4 types of kernels: Linear, Radial, Polynomial and Sigmoid (tuning parameter γ)

One vs One (OvO)

Suppose we have k classes

- Build $\binom{k}{2}$ classifiers for each pair of classes
- Generate predictions for each classifer and take majority

ex. suppose $k = 4$

Classifier	Prediction
1 vs 2	2
1 vs 3	1
1 vs 4	4
2 vs 3	2
2 vs 4	2
3 vs 4	3

$\rightarrow$ **Predict Class 2**

Note: Default in e1071 library in R for multi-class SVMs

One vs All (OvA)

- Build k classifiers. Compare each class with everything else.
- Generate predictions for each classifer and take maximum decision value $(\beta_0 + \boldsymbol{x}'\boldsymbol{\beta})$

ex. $k = 4$

Classifier	Decision Value
1 vs Not1	-3.01
2 vs Not2	0.01
3 vs Not3	1.01
4 vs Not4	0.87

$\rightarrow$ **Predict Class 3**

Divide by 2 (DB2)

Vural and Dy (2004)[1] suggest splitting the classes into 2 subsets until each subset has a single class.

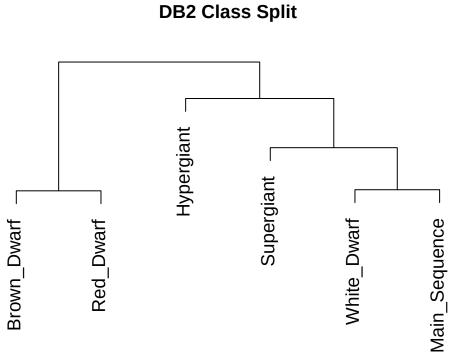
- "Tree method" $\rightarrow$ Build a decision tree that separates classes.
- Classifier for each split $\rightarrow k - 1$ classifiers

ex. Using star data ($k = 6$):

- Is this a dwarf star or giant/main sequence? (1)
- If it's a Dwarf:
  - Is this a brown/red dwarf or white dwarf? (2)
  - If not white: is this brown dwarf or red dwarf? (3)
- If it's not a dwarf:
  - Is this a giant star or main-sequence? (4)
  - If it's a giant star: is it super- or hyper-? (5)

DB2 data splits

- Splitting the classes is left to the user.
- One idea is to do hierarchical clustering on class centroids.
- Seek to find "even" splits → minimizes average # of classifiers needed to get to a single class (Ward's linkage)



Results

- Split data into train (75%) and test sets.
- Using 10-fold cross validation, do a grid search on tuning parameters (C, γ, and the kernel) using training set.
- Predict accuracy on test set

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%**

Star dataset is very separable. DB2 and OvO do the best.

Confusion Matrix (Test set)

Truth (top) vs predicted (left)

Class	BD	RD	WD	MS	SG	HG
Brown Dwarf	13	1	0	0	0	0
Red Dwarf	1	7	0	0	0	0
White Dwarf	0	0	7	0	0	0
Main-Seq	0	0	0	11	0	0
Supergiant	0	0	0	0	9	0
Hypergiant	0	0	0	0	0	11

SVM only has "trouble" with separating Brown and red dwarfs.

Simulation

We wish to see how the 3 methods perform under levels of:

1. Class Separation (Low, Med, High)
2. Class Balances (Balanced, Varied, Imbalanced)

Simulation Plan

- Simulate 240 datapoints with Balanced, Varied, and imbalanced class counts.
  - Varied = (20,40,50,60,40,30)
  - Unbalanced = (12, 20, 50, 110, 30, 18)
- Give each group its own mean for 3 covariates. Change separation (Low, Med, High) by scaling these means.
  - Scale group means by 0.5 (Low), 1 (Moderate) and 2 (High)
  - I used:
    - Covariate 1: Each group mean $\sim$ N(0,1)
    - Covariate 2: (0,0,-.2, 1, 1, 3)
    - Covariate 3: (-.5, 0, 0, 1, 2, 2)
    - Covariate 4: N(0,1) noise
    - Normal data, $\sigma = 1$
- 9 Datasets total. Keep the same data across 3 methods and test performance.

Simulation Results

(a) OvO Test set accuracy

OvO	Bal	Varied	Unbal
Low	33.3%	55.0%	51.7%
Moderate	50.0%	66.7%	71.7%
High	75.0%	80.0%	86.7%

(b) OvA Test set accuracy

OvA	Bal	Varied	Unbal
Low	45.0%	45.0%	58.3%
Moderate	59.7%	58.3%	74.6%
High	63.3%	78.3%	90.0%

(c) DB2 Test set accuracy

DB2	Bal	Varied	Unbal
Low	40.0%	45.0%	65.0%
Moderate	60.0%	68.3%	76.7%
High	73.3%	78.3%	78.3%

Green=Best, Orange=Worst by 2%

- Higher accuracy for unbalanced classes → predicting everything in common classes
  - All methods struggle to predict well in rare classes
- Mixed bag results → all have utility?
- High Separability, balanced classes → DB2 and OvA do well
- OvO and DB2 were actually faster. Most classifiers do not need the full data.

Class prediction accuracy example

Table: 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*

Maybe a result of a low sample size?
Rare classes have super low accuracies. (Overall 74.6% for unbalanced)

Conclusion

- DB2 and OvO performed best on star dataset
  - See how it performs on thousands of stars
- Mixed bag of results for simulation. All methods have some utility
- Rare classes were hard for all 3 methods.
  - Future work: Increasing sample size?

Final thoughts

1. One vs One $\binom{k}{2}$
   - Easy to implement (existing software)
   - Blows up as k gets larger. If $k = 20 \rightarrow 190$ Classifiers
2. One vs All (k)
   - Works well for unbalanced data?
   - Slowest for small k. All classifiers train on all the data
3. Divide by 2 (k-1)
   - Fast and requires the least number of classifiers
   - Requires specification of class divisions
   - Took longer to implement

These can be generalized to any binary classifier!

references

1. 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