Data, Environment and Society: Lecture 27: Support Vector Machines, continued

Instructor: Duncan Callaway
GSI: Seigi Karasaki

November 27, 2018

Announcements

- HW10 due today.
- HW11: Optional, for extra credit.
- Seigi and I will be at lab next week
  - Attendance optional – but we will help with projects.
- Final lecture, here on Tuesday 12/4.
  - Covers neural nets and course wrap-up
  - But I'll also post a video
  - No reading for this.
- Projects: Due December 11 at 6pm.

A few more project notes:

- Don't forget to normalize your data if you're using lasso, ridge, elastic net.
- It's important to get your resource allocation right, but more important to run some models.
- That is, don't let the perfect be the enemy of the good!
- We'll be in Barrows 110 next monday 10-12 for consultations on your projects.

Objectives for today

1. Review fundamentals building up to SVM (from two weeks ago!)
   - Hyperplanes
   - Maximal margin hyperplanes for linearly separable data
   - Support vector classifiers: linear boundaries for non-separable data
2. Open your horizons to the **support vector machine**
   - This provides **nonlinear** separations of the feature space
3. Discuss the basic pipeline for model building with cross validation
   - Train, test and validation data sets
   - Fit the model
   - Choose "hyperparameters"
   - score the model
4. Course reviews

Theoretical example: Classification with hyperplanes

Suppose we have

- blue points
- red points

A "separating hyperplane" has the property that:

Using the plane for predictions

This part is simple. If we have a test observation, we simply evaluate $f(x_{\text{test}})$ and assign it to a class on the basis of the sign of the result.

Maximal margin hyperplane (MMH)

- MMH defined: The MMH is the hyperplane that *maximizes* the distance to the closest point.
  - In other words: it is the plane that is farthest from the data.
- Important: This requires that the data are *linearly separable*.
- 2-3 points defining the hyperplane are termed "support vectors" because each observation is a "vector" of information that supports the plane.

Support vector classifier details

We'll solve a slightly different optimization problem.

- Still classifies observations on the basis of what side of the hyperplane they're on
- But a few observations can be mis-classified.
- But with C, the ϵ values are chosen within a budget.
  - $\epsilon_i = 0$, no margin violation
  - $0 < \epsilon_i < 1$, in the margin but on the correct side of the plane
  - $\epsilon_i > 1$, on the wrong side of the plane.

The optimization problem is:

Setup for SVM "Kernels": Linear boundary

What's the relationship between α and β?

Let $\langle x_i, x_k \rangle = \sum_{j=1}^{p} x_{ij} x_{kj}$

One can show that:

$$\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \Leftrightarrow \beta_0 + \sum_{k=1}^{n} \alpha_k \langle x_i, x_k \rangle$$

$$f(x_i) = \beta_0 + \sum_{k \in \mathbb{S}} \alpha_k \langle x_i, x_k \rangle$$

Decision boundaries aren't always so simple...

What if the boundary looked like this?

We might get better performance if we replaced the linear plane in the optimization problem with something else.

SVM Kernels

Define a generic model:

$$f(x_i) = \beta_0 + \sum_{k=1}^{p} \alpha_k K(x_i, x_k)$$

With kernels:

$$K_{\text{linear}}(x_i, x_k) = \langle x_i, x_k \rangle = \sum_{j=1}^{p} x_{ij} x_{kj}$$

$$K_{\text{polynomial}}(x_i, x_k) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^d$$

$$K_{\text{radial}}(x_i, x_k) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{kj})^2)$$

The radial kernel serves to give training observations far from a test point less weight (due to negative exponential).

For the radial kernel, all the data comprise the model – analogous to KNN.

Polynomial kernel expanded

$$K_{\text{polynomial}}(x_i, x_k) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^d$$

$$d = 1 \Rightarrow K_{\text{polynomial}}(x_i, x_k) =$$

$$d = 3 \Rightarrow K_{\text{polynomial}}(x_i, x_k) =$$

...So adjusting d is essentially "feature engineering" to manipulate what order terms you put in your classifier.

Radial Basis Functions

$$K_{\text{radial}}(x_i, x_k) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{kj})^2),$$

$$f_{\text{radial}}(x_i) = \beta_0 + \sum_{k=1}^{p} \alpha_k K_{\text{radial}}(x_i, x_k)$$

Observations k in the training set that are close to observation i will get...

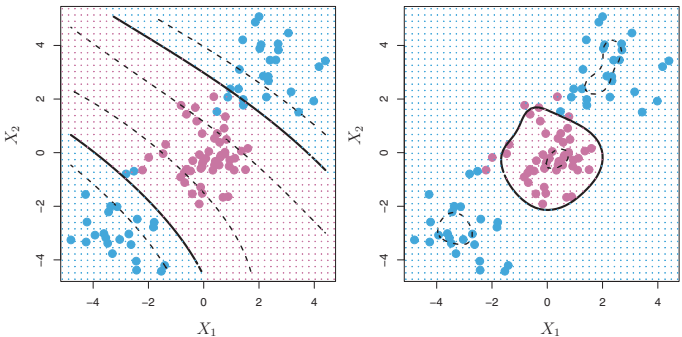
- More weight?
- Less weight?

Assume α is positive. Then points closer to x_i will yield

- Larger f?
- Smaller f?

Larger evaluations of the kernel push f in the positive direction $\rightarrow$ tend to classify with the positive variable.

What do the kernels look like?



Left: Polynomial, $d = 3$.
Right: Radial.

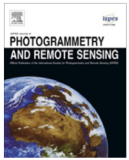
Note that d and γ are parameters to tune (via cross validation!)

SVM Example

Contents lists available at SciVerse ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs



Comparison of support vector machine, neural network, and CART algorithms for the land-cover classification using limited training data points

Yang Shao [a,*], Ross S. Lunetta [b]

[a] US Environmental Protection Agency, National Research Council, National Exposure Research Laboratory, 109 T.W. Alexander Drive, Research Triangle Park, NC 27711, USA
[b] US Environmental Protection Agency, National Exposure Research Laboratory, 109, T.W. Alexander Drive, Research Triangle Park, NC 27711, USA

Shao and Lunetta setup

Loads of remote sensing data (MODIS):

- 46 input features for each 250 m^2 pixel
  - 23 short wave infrared (SWIR) surface reflectance
  - 23 Enhanced Vegetation Index metrics – basically a summary of the wavelengths
- Training data from National Land Cover Dataset (NLCD), classifying land as
  - urban,
  - deciduous forest,
  - evergreen forest,
  - agricultural land, and
  - wetland
- Question: How do SVM, Neural networks (coming soon!), and regression trees perform relative to one another?

Shao and Lunetta SVM Result

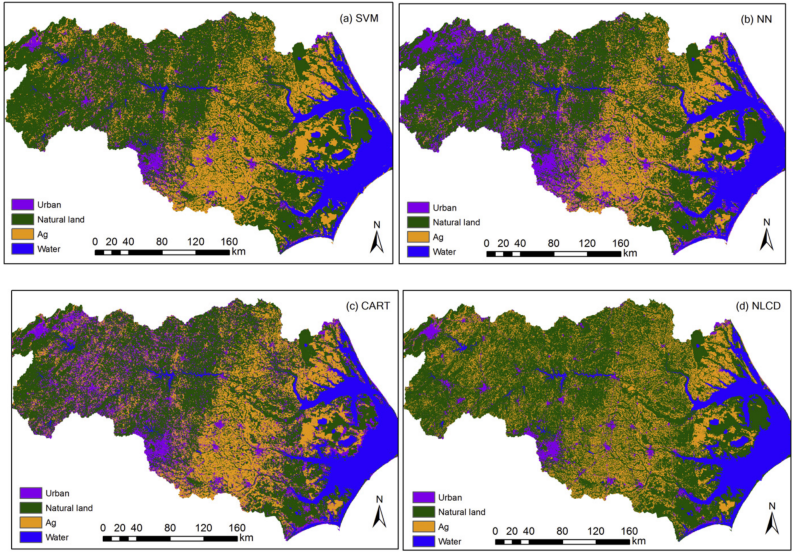
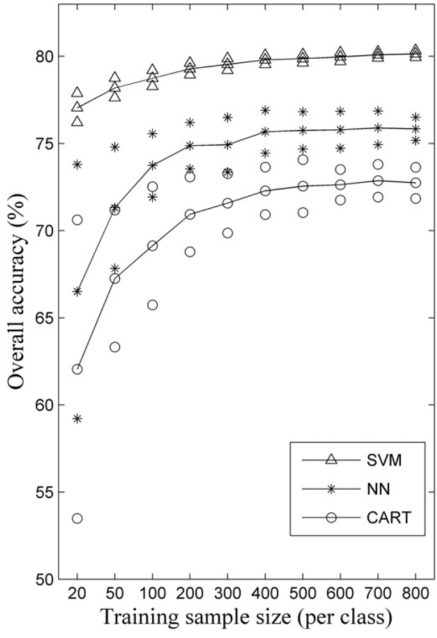


Fig. 3. Comparison of classification results for SVM (a), NN (b), and CART (c) algorithms. The NLCD 2001 (d) is also included as reference.


Overall accuracy (%)
80
75
70
65
60
55
50
20 50 100 200 300 400 500 600 700 800
Training sample size (per class)
SVM
NN
CART

Textbook example: Heart Data

First: Receiver operating characteristic (ROC)

Sensitivity = the fraction of "positives" that are correctly identified as positives

Specificity = fraction of negatives that we correctly identify as negatives

True positive rate = sensitivity

False positive rate = $1-$ specificity
Ideal classifiers have large true positive rate and low false positive.

For a given method, there are usually parameters one can tune to explore the tradeoff between true and false positives.

Classifying heart disease

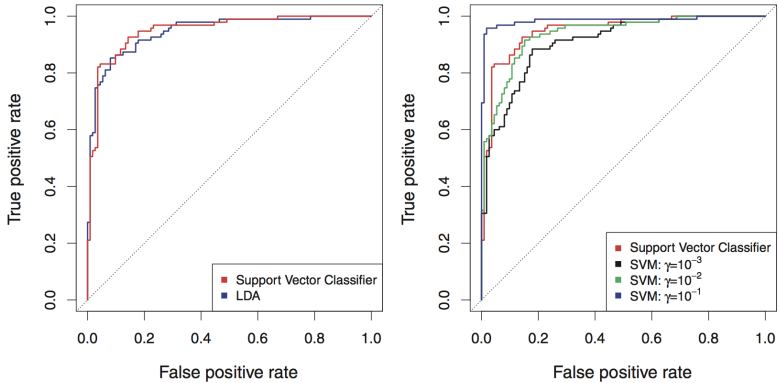


FIGURE 9.10. *ROC curves for the* `Heart` *data training set.* Left: *The support vector classifier and LDA are compared.* Right: *The support vector classifier is compared to an SVM using a radial basis kernel with* $\gamma = 10^{-3}$, 10^{-2}, *and* 10^{-1}.

Classifying heart disease – test data

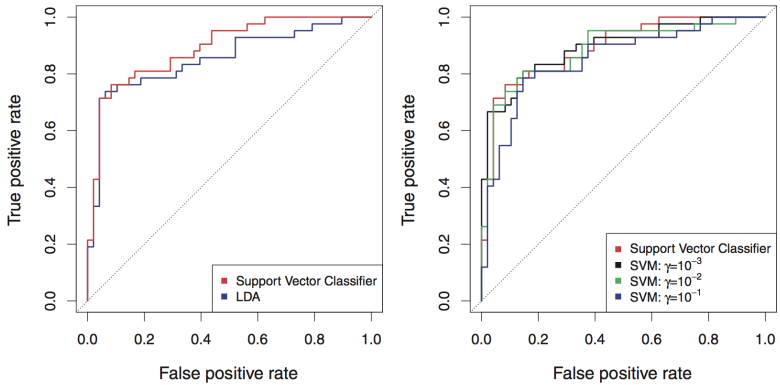


FIGURE 9.11. *ROC curves for the test set of the* `Heart` *data.* Left: *The support vector classifier and LDA are compared.* Right: *The support vector classifier is compared to an SVM using a radial basis kernel with* $\gamma = 10^{-3}$, 10^{-2}, *and* 10^{-1}.

Interpreting Heart data result

On training data, SVM with radial kernels are exceptional

...But not so much on test data. Why?

Interpreting Heart data result

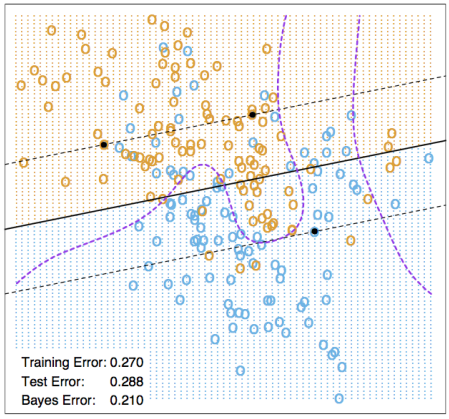
On training data, SVM with radial kernels are exceptional

...But not so much on test data. Why?

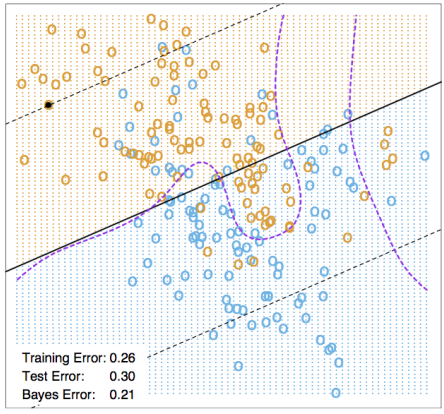
The decision boundary is really "wiggly" and prone to overfit.

It appears that SVC would have lower variance / higher bias and they balance perfectly in this particular case.

Speaking of wiggly boundaries: From Elements of Statistical Learning



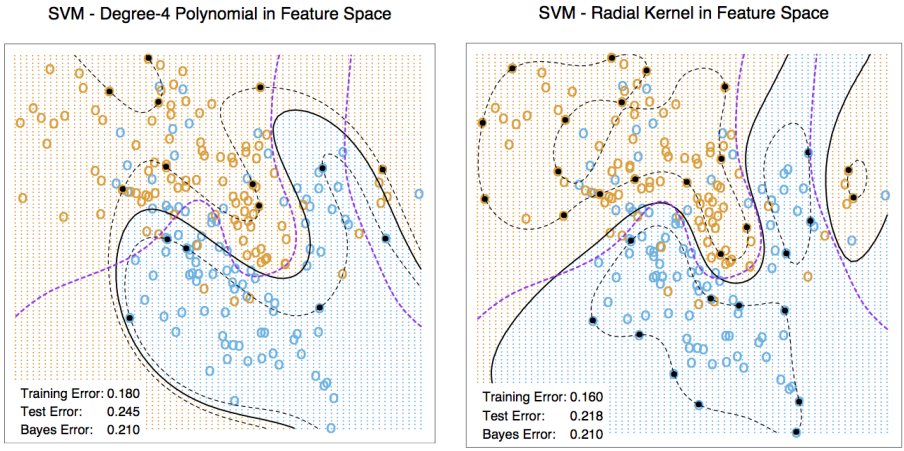
$C = 10000$



$C = 0.01$

These boundaries constructed with SVC. Purple is "truth" (what they used to generate the data). Note! In ESL C is the inverse of C in ISLR. So large C here corresponds to small C there, and vice versa.

One more example: From Elements of Statistical Learning



These boundaries constructed with SVM. Purple is "truth" (what they used to generate the data).

Models have two different classes of parameters

1. Parameters that enter as decision variables for minimizing loss function:

$$\text{Shrinkage: } \beta^* = \arg\min_{\beta} \sum_{i=1}^{N} (Y_i - X_i\beta)^2 + \lambda \cdot R(\beta)$$

$$\text{CART: } \{j^*, s^*\} = \arg\min_{j \in J, s \in X_j} \sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

2. "Hyperparameters": parameters that constrain how you solve the loss function. These generally prevent overfit.

Models have two different classes of parameters

1. Parameters that enter as decision variables for minimizing loss function:

$$\text{Shrinkage: } \beta^* = \arg\min_{\beta} \sum_{i=1}^{N} (Y_i - X_i\beta)^2 + \lambda \cdot R(\beta)$$

$$\text{CART: } \{j^*, s^*\} = \arg \min_{j \in J, s \in X_j} \sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

2. "Hyperparameters": parameters that constrain how you solve the loss function. These generally prevent overfit.
   - λ in shrinkage methods
   - How deep to grow a classification tree?
   - $\sum_{i=1}^{n} \epsilon_i$ in SVM

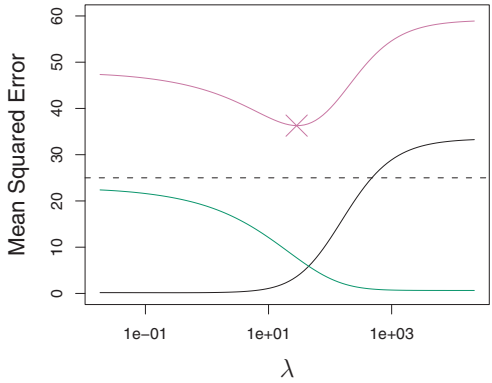
Ways to minimize the loss function

- Closed form solution – e.g. normal equations
- Gradient search

In both cases, we're relying on the condition that the gradient of the loss function approaches zero as we approach the optimal solution

Ways to choose hyperparameters

- **Grid search**: This is what we've done with shrinkage methods, when there is just one parameter to tune (λ)
- **Randomized parameter search**: This is what you're doing in HW10. Works well when you have lots of hyperparameters to tune.



In both cases, all we're doing is

- Creating a list of candidate hyperparameters (or sets of hyperparameters)
- Training the model (with the training data) for each hyperparameter in the list
- Choosing the hyperparameter with the best cross-validated error.

Zoom out: Train, (cross) validate and test

Using slightly different language, some definitions from from Brian Ripley, Pattern Recognition and Neural Networks, 1996, page 354:

- **Training set**: "A set of examples used for learning, that is to fit the parameters of the classifier."
- **Validation set**: "A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network."
- **Test set**: "A set of examples used only to assess the performance of a fully-specified classifier."

I'll use these definitions going forward.

Zoom out: Train, (cross) validate and test

Using slightly different language, some definitions from from Brian Ripley, Pattern Recognition and Neural Networks, 1996, page 354:

- **Training set**: "A set of examples used for learning, that is to fit the parameters of the classifier."
  - For minimizing the loss function
- **Validation set**: "A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network."
- **Test set**: "A set of examples used only to assess the performance of a fully-specified classifier."

I'll use these definitions going forward.

Zoom out: Train, (cross) validate and test

Using slightly different language, some definitions from from Brian Ripley, Pattern Recognition and Neural Networks, 1996, page 354:

- **Training set**: "A set of examples used for learning, that is to fit the parameters of the classifier."
  - For minimizing the loss function
- **Validation set**: "A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network."
  - For choosing hyperparameters.
- **Test set**: "A set of examples used only to assess the performance of a fully-specified classifier."

I'll use these definitions going forward.

Zoom out: Train, (cross) validate and test

Using slightly different language, some definitions from from Brian Ripley, Pattern Recognition and Neural Networks, 1996, page 354:

- **Training set**: "A set of examples used for learning, that is to fit the parameters of the classifier."
  - For minimizing the loss function
- **Validation set**: "A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network."
  - For choosing hyperparameters.
- **Test set**: "A set of examples used only to assess the performance of a fully-specified classifier."
  - For a final check – no more model fitting allowed here!

I'll use these definitions going forward.

Why use validation *and* test sets?

I.e., why isn't it enough to use just a validation set?

Why use validation *and* test sets?

I.e., why isn't it enough to use just a validation set?

- Because we iteratively tune hyperparameters with the validation sets, the models "see" the data
- So in essence the validation sets *are* training the model
- Keeping a test set locked away until the end of the model fitting process is the only way to be sure the comparison of your models is fair.
- We haven't done this in the course – but if you wish to compare different types of models (e.g. SVM vs random forest) then it's good practice.