Statistical Machine Learning 16 — SVM Kernelization & Boosting October 22, 2015

Lecturer: Prof. Homrighausen Scribe: Veronica Burt

1 Previous Lectures

We have been discussing

- Support Vector Machines to find the optimal separating hyperplane.
- How linear boundaries are not always optimal, but we can use Kernelization to create a good linear boundary in a different space.
- How to derive the support vector classifier via penalized loss methods.

2 Surrogate Losses

- The SVM is defined
  $\min_{\beta,beta_0} \sum_{i=1}^{n} [1 - Y_i h(X_i)]_+ + \tau ||\beta||_2^2$,
  where $[1 - Y_i h(X_i)]_+$ is the hinge loss and $\tau ||\beta||_2^2$ is the complexity term.
- Similar to linear regression, we may want to minimize $\sum_{i=1}^{n} \mathbf{1}(Y_i \neq \hat{g}(X_i)) + \tau ||\beta||_2^2$. However, this is nonconvex in $u = h(X)Y$, so it is not efficiently solvable. To fix this, we can find a nearby convex problem to approximate it, known as convex relation with a surrogate loss function.
- We can use a surrogate loss function that mimics this fuction, but is still convex. We've already done this:
  Hinge: $[1 - Yh(X)]_+$
  Logistic: $log(1 + e^{-Yh()X})$
- We know that the Logistic Regression likelihood is $YX^T\beta - log(1 + e^{X^T\beta})$. To turn this into a quantity we want to minimize, we simply flip the sign:
  $-YX^T\beta + log(1 + e^{X^T\beta})$ $=log(\frac{1+e^{h(X)}}{e^{Yh(X)}})$ $=log(e^{-Yh(X)} + e^{h(X)-Yh(X)})$ $=log(e^{-Yh(X)} + e^{(1-Y)h(X)}$
- See Figure 1 for a visualization of the losses.

3 SVMs in Practice

- We can find the basic SVM functions in the C++ library libsvm. The R package e1071 calls the libsvm library.
- The path algorithm svmpath uses the penalized likelihood method.
- Figure 2 shows output of the R package SVM.

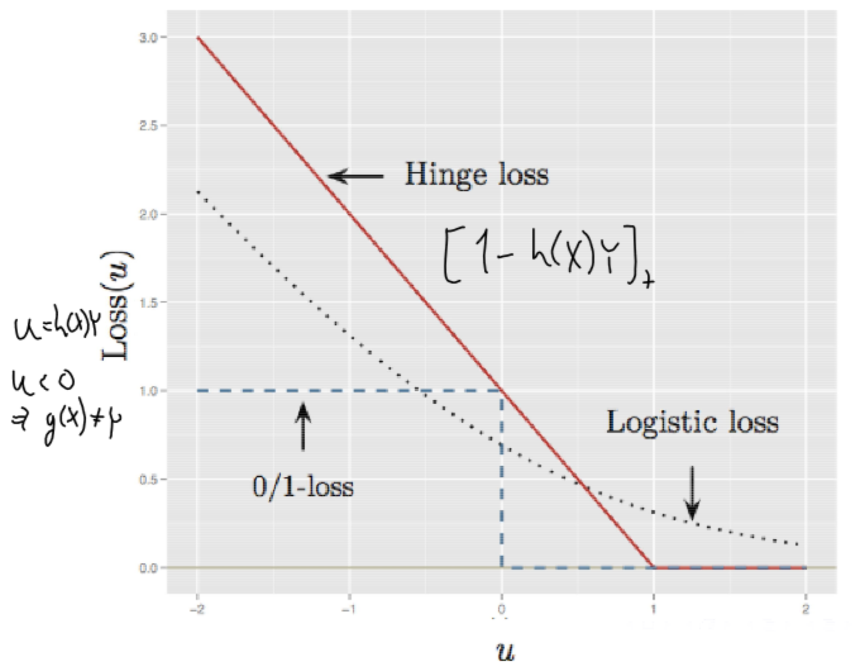


Figure 1: The Logistic loss is infinitely differentiable, while the Hinge loss has a point of discontinuity (the hinge).

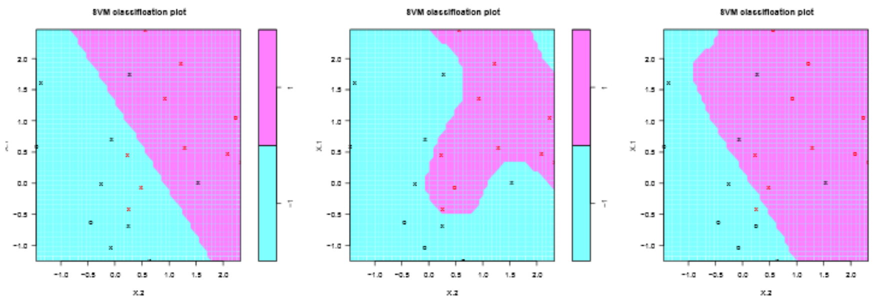


Figure 2: The X's represent support vectors and the O's represent nonsupport vectors. The black and red colors indicate the classes. Each of the boundaries are linear in the kernel space, but the right two examples are nonlinear in the space of our data.

4 Multiple Classification

- Sometimes, we have more than two categories in our classification problem. This is known as multiclass classification.
- There are two main approaches:

  One-versus-one

  One-versus-all
- If we have G possible classes, for the one-versus-one approach we run $G(G-1)/2$ possible pairwise classifications. For a given test point X, we find $\hat{g}_k(X)$ for $k = 1, ..., G(G-1)/2$ fits. The result is a vector $\hat{G} \in \mathbb{R}^G$ that contains the total number of times X was assigned to each of the G classes. We report $\hat{g}(X) = \text{argmax}_g \hat{G}$, the class that was assigned the most times. This approach uses all of the class information, but can be slow. Typically it is only run on a subset of the data.
- If we have G possible classes, for the one-versus-all approach we fit G SVMs by collapsing the remaining $G-1$ subsets together (for example, compare class 2 to not 2). Take all $\hat{h}_g(X)$ for $g = 1, ..., G$, where class g is coded as 1 and all other classes are coded as -1. Assign $\hat{g}(X) = \text{argmax}_g \hat{h}_g(X)$. We can easily use all of the data for this approach, but we don't use all of the class information at once.
- Note that these strategies can be applied to any classifier.

5 Boosting Overview

- Remember from previous lectures that Bagging is a procedure that takes a low bias, high variance estimator and attempts to reduce the risk via averaging. However, this isn't effective if the things being averaged are highly correlated.
- Boosting has a similar philosophy- it is a procedure that takes a high bias, low variance estimator and attempts to reduce the bias.
- Bagging aggregates over many independent bootstrap draws (while the trees are correlated, the samples are drawn independently).
- Boosting makes the algorithm pay more attention to the poorly classified observations: the procedure finds observations that are poorly classified, up-weights these observations, and then trains a new classifier.

6 Boosting for Regression

- There are three main components to boosting:

  1. A base learner $\hat{f}$. This is often an uncomplicated estimator, such as a spline. It is often an easy to calculate, high bias, low variance method. Any complexity parameters for f are commonly set at a fixed, low number of parameters. For example, in regression trees, we can cap the number of interactions included in $\hat{f}$.
  2. A learning rate λ. This can be data dependent.
  3. The number of base learners B. This acts like the number of iterations
- Trees are great example of a base learner because they have low bias but high variance, which is great for boosting.

- To boost a regression tree:

  1. Set $\hat{f} \equiv 0$, and $R = Y \in \mathbb{R}^n$. In the case of regression, think of R as the residuals from our model.

  2. Fix the tree complexity M and the learning rate λ. Typically, we let M be fairly small. We can think of λ as the amount of the new tree we want to consider.

  3. For $b = 1, ..., B$: Fit $\hat{f}_b$ with $M+1$ regions to $\tilde{\mathcal{D}} = \{(X_1, R_1), ..., (X_n, R_n)\}$. Update: $\hat{f} \leftarrow \hat{f} + \lambda \hat{f}_b$, and $R \leftarrow R - \hat{f}$.

- This process upweights observations with high residuals.
- The output is $\hat{f} = \sum_{b=1}^{B} \lambda \hat{f}_b$
- This is an additive model; we stitched together a bunch of classifiers with low variance.
- We can think of λ as the length of steps we need to take; a smaller λ means a larger required B. If λ is too large, our steps are too large, and we will overshoot the optimal solution.
- In practice, B is set via cross-validation or other risk estimates. Boosting is insensitive to overfitting by choosing B too large. λ is set to a small level, such as $\lambda = .01$ or let λ_b to converge to zero, polynomially or exponentially. We don't want to choose λ via cross-validation because 2-dimensional cross validations grids are complicated.

7 Local Averaging

- Our goal is to develop a prediction function $\hat{f} : \mathbb{R}^p \to \mathbb{R}$ for predicting Y given an X.
- Commonly, $\hat{f} = X^T \beta$, which is constrained linear regression.
- This greatly simplifies algorithms, without sacrificing too much flexibility.
- Directly modeling the nonlinearity can be more natural.
- We have been modeling the Bayes Rules, $\mathbb{E}[Y|X]$ and $\text{argmax}_g \mathbb{P}(Y = g|X)$.
- Estimating expectations is easy: if $Y_1, Y_2, ..., Y_n$ all have expectations μ, then $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} Y_i$ is the intuitive estimator of μ, and a reasonable prediction of a new Y.
- Similarly, we can estimate $\mathbb{E}[Y|X]$ with an indicator function: $\hat{f}(X) = \frac{1}{n_x} Y_i \mathbf{1}(X_i = X)$, where $n_x = \sum_{i=1}^{n} \mathbf{1}(X_i = X)$. The indicator creates a conditional expectation, where we only consider features exactly equal to the features we are interested in.
- However, there are almost never any X_i at X! We can instead relax this constraint and consider observations that are nearby.
- Suppose we have data $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$. Then,
  $\hat{f}(X) = \frac{1}{n_x} \sum_{i=1}^{n_x} Y_i \mathbf{1}(||X_i - X|| \leq t)$, where $n_x = \sum_{i=1}^{n} \mathbf{1}(||X_i - X|| \leq t)$.
- The tuning parameter t quantifies closeness. We could also weight the observations and make the weights go exponentially to zero as points get farther away, rather than using an indicator function.
- See Figures 3 and 4 for some examples of local averaging.
- An unbiased estimate only occurs at $X_i = X$, but it's a poor estimator due to high variance.

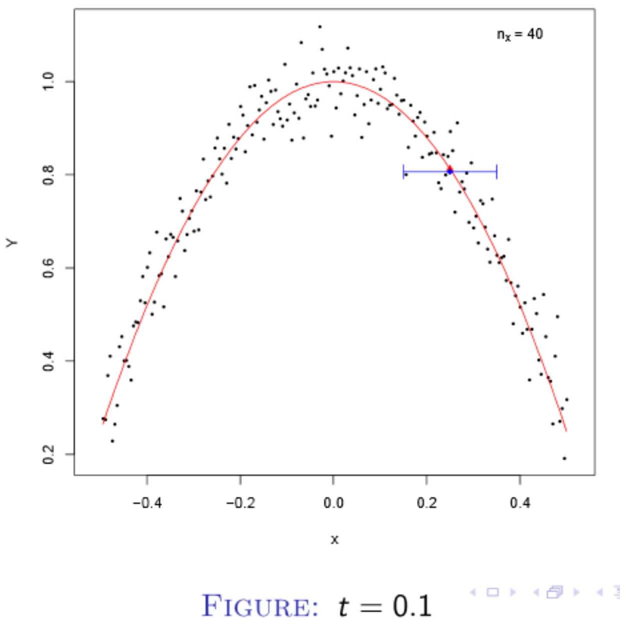


Figure 3: An example of local averaging where t = .1.

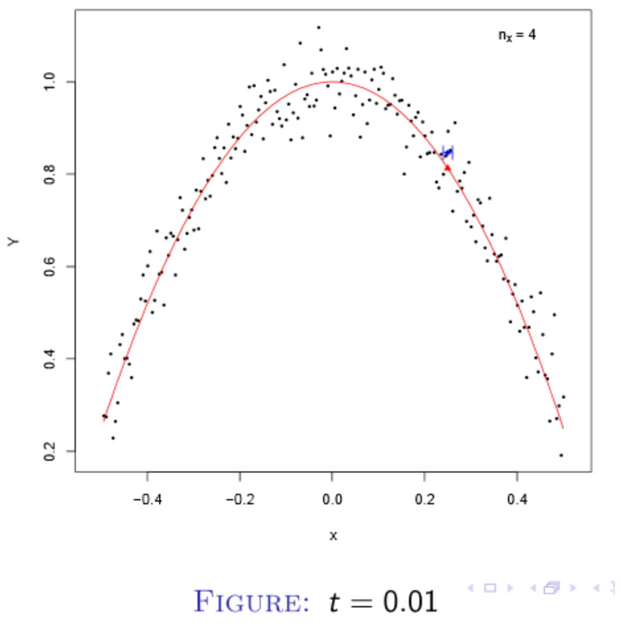


Figure 4: An example of local averaging where t = .01.

- We can't always fit a model like this. It works best if p is small relative to n, and we specify t properly. As p gets large, no observations are nearby, and all of our predictions turn into extrapolations, since no data points will be in our prediction space.
- This is the Curse of Dimensionality.

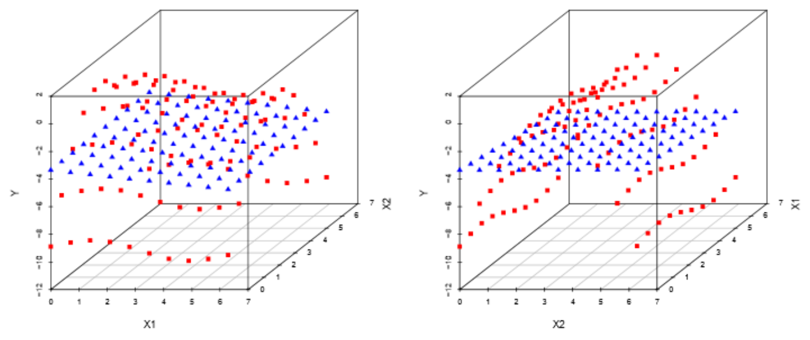
8 Curse of Dimensionality

- Fix the dimension of p (as an even number). Let S be a hypersphere with radius r and let C be a hypercube with side length $2r$.
- Then, the volume of S and C are $V_S = \frac{r^p \pi^{p/2}}{(p/2)!}$ and $V_C = (2r)^p$.
- The volume of the hypersphere goes to zero quickly since the factorial in the denominator races to infinity faster than the exponents in the numerator. The diagonal length of the hypercube is always proportional to $\sqrt{p}$, and the volume goes to zero. This makes the hypercube jagged.
- The ratio of the volumes of a circumscribed hypersphere by a hypercube is $\frac{V_C}{V_S} = \frac{(2r)^p * (p/2)!}{r^p \pi^{p/2}} = (\frac{4}{\pi})^d d!$, where $d = p/2$.
- This means all of the volume in the hypercube is in the corners, meaning that our observations are all in the corners. Also, there are many corners, so it is likely that there is only one observation per corner. This makes our observations isolated in their respective corners, and predicting for observations outside of these corners is extrapolation. We would need a very large number of points to fit this well.

9 Additive Models

- We can combine linear and nonlinear models to create a flexible model that somewhat compensates for the problem of too many dimensions; $f(X) = f_1(x_1) + ... + f_p(x_p) = \sum_{j=1}^{p} f_j(x_j)$
- This is just slightly more complicated than estimating a fully linear model, since all of our inputs enter the estimator separately. This algorithmic approach is known as backfitting. It is a version of gradient descent.
- Additive model are usually defined using expectations, which are replaced with empirical expectations in analysis.
- The update is a Gauss-Siedel update, which is an iterative method for solving linear, square systems. It is a cyclic update that converges to the underlying regression function, $\mathbb{E}[Y|X]$, by taking a large number of iterations. For $j = 1, ..., p, 1, ..., p, 1, ...$

  $f_j(x_j) \leftarrow \mathbb{E}[Y - \sum_{k \neq j} f_k(x_k)|x_j]$
- To backfit for additive models, choose a univariate nonparametric smoother $\mathcal{S}$ and form all marginal $\hat{f}_j$. Typically, S is something simple like a cubic smoothing spline, or a local average. Iterate over j until the algorithm converges:
  1. Define the residuals $R_i = Y_i - \sum_{k \neq j} \hat{f}_k(X_i^k)$
  2. Smooth the residuals $\hat{f}_j = \mathcal{S}(R)$
  3. Center $\hat{f}_j \leftarrow \hat{f}_j - n^{-1} \sum_{i=1}^{n} \hat{f}_j(X_i^j)$
- Repeat the process until the residuals don't change much. Report $\hat{f}(X) = \bar{Y} + \hat{f}_1(x_1) + ... + \hat{f}_p(x_p)$
- See the Lecture slides for example R code. For interacting 3D plotting, use the package rgl.



(These are the fitted values only. Red squares: GAM, Blue triangles: multiple linear regression)

Figure 5: The fitted values from the simulation. The red squares represent the fitted values from a generalized additive model, and the blue triangles represent the fitted values from multiple linear regression. Note that the multiple linear regression does not accurately fit the 'taco' shape of the data.

- See Figure 5 for a visualization of an additive model verses a multiple regression model.