Support vector machines and kernelization

-Statistical Learning and Data Mining-

Lecturer: Darren Homrighausen, PhD

Kernel methods

Intuition: Many methods have linear decision boundaries

We know that sometimes this isn't sufficient to represent data

Example: Sometimes we need to included a polynomial effect or a log transform in multiple regression

Sometimes, a linear boundary, but in a different space makes all the difference..

Optimal separating hyperplane

Reminder: The Wolfe dual, which gets maximized over α, produces the optimal separating hyperplane

$$\text{Wolf dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k Y_i Y_k X_i^\top X_k$$

(this is all subject to $\alpha_i \geq 0$)

A similar result holds after the introduction of slack variables

(e.g. support vector classifiers)

Important: The features only enter via

$$X^\top X' = \langle X, X' \rangle$$

(Kernel) ridge regression

Reminder: Suppose we want to predict at X, then

$$\hat{f}(X) = X^\top \hat{\beta}_{\text{ridge},\lambda} = X^\top \mathbb{X}^\top (\mathbb{X}\mathbb{X}^\top + \lambda I)^{-1} Y$$

Also,

$$\mathbb{X}\mathbb{X}^\top = \begin{bmatrix} \langle X_1, X_1 \rangle & \langle X_1, X_2 \rangle & \cdots & \langle X_1, X_n \rangle \\ & \vdots & & \\ \langle X_n, X_1 \rangle & \langle X_n, X_2 \rangle & \cdots & \langle X_n, X_n \rangle \end{bmatrix}$$

and

$$X^\top \mathbb{X}^\top = [\langle X, X_1 \rangle, \langle X, X_2 \rangle, \cdots, \langle X, X_n \rangle]$$

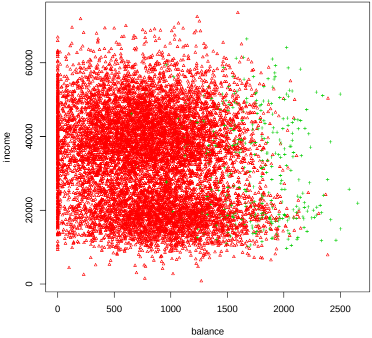
Again, we have the covariates enter only as

$$\langle X, X' \rangle = X^\top X'$$

Logistic regression: transformations

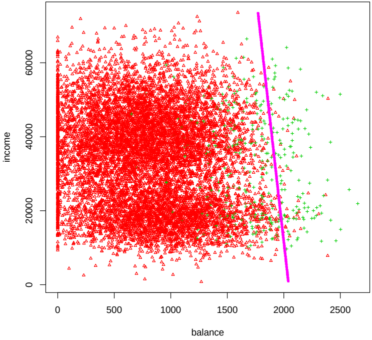
Let's look at the default data in "Introduction to Statistical Learning"

In particular, we will look at default status as a function of balance and income



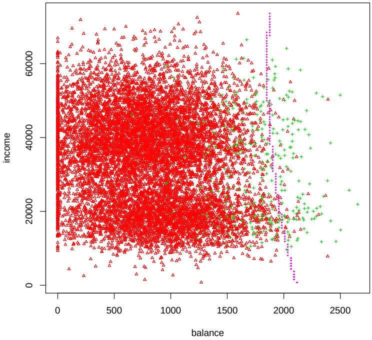
Logistic regression: transformations

```
out.glm  = glm(default~balance + income,family='binomial')
```



Logistic regression: transformations

```
out.glm  = glm(default~balance + income +
                    I(income^2),family='binomial')
```



Conclusion: A Linear rule in a transformed space can have a nonlinear boundary in the original features

Logistic regression: transformations

Reminder: The logistic model: untransformed

$$\begin{aligned}\text{logit}(\mathbb{P}(Y = 1|X)) &= \beta_0 + \beta^\top X \\ &= \beta_0 + \beta_1 \text{balance} + \beta_2 \text{income}\end{aligned}$$

The decision boundary is the hyperplane $\{X : \beta_0 + \beta^\top X = 0\}$

This is linear in the feature space

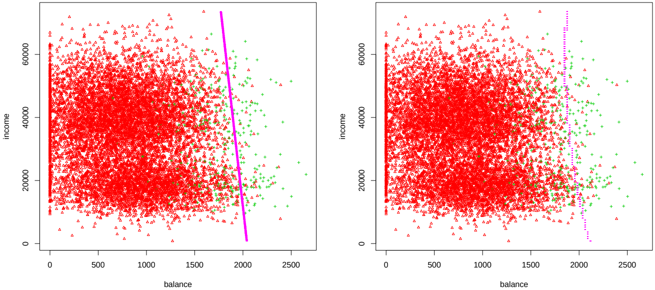
Logistic regression: transformations

Adding the polynomial transformation $\Phi(X) = (x_1, x_2, x_2^2)$:

$$\begin{aligned}\mathrm{logit}(\mathbb{P}(Y = 1|X)) &= \beta_0 + \beta^\top \Phi(X) \\ &= \beta_0 + \beta_1 \mathrm{balance} + \beta_2 \mathrm{income} + \beta_3 \mathrm{income}^2\end{aligned}$$

Decision boundary is still a hyperplane $\{X : \beta_0 + \beta^\top \Phi(X) = 0\}$

This is nonlinear in the feature space!



Logistic regression: transformations

Of course, as we include more transformations,

- We need to choose the transformations manually
- Computations can become difficult if we aren't careful
  (Example: Solving the least squares problem takes something like np^2 computations)
- We need to regularize to prevent overfitting

Can we form them in an automated fashion?

Kernel Methods

Nonnegative definite matrices

Let $A \in \mathbb{R}^{p \times p}$ be a symmetric, nonnegative definite matrix:

$$z^\top A z \geq 0 \text{ for all } z \quad \text{and} \quad A^\top = A$$

Then, A has an eigenvalue expansion

$$A = UDU^\top = \sum_{j=1}^{p} d_j u_j u_j^\top$$

where $d_j \geq 0$

Observation: Each such A, generates a new inner product

$$\langle z, z' \rangle = z^\top z' = z^\top \underbrace{I}_{\text{Identity}} z'$$

$$\langle z, z' \rangle_A = z^\top A z'$$

(If we enforce A to be positive definite, then $\langle z, z \rangle_A = ||z||_A^2$ is a norm)

Nonnegative definite matrices

Suppose A_i^j is the (i,j) entry in A, and A_i is the i^{th} row

$$Az = \begin{bmatrix} A_1^\top \\ \vdots \\ A_p^\top \end{bmatrix} z = \begin{bmatrix} A_1^\top z \\ \vdots \\ A_p^\top z \end{bmatrix}$$

Note: Multiplication by A is really taking inner products with its rows.

Hence, A_i is called the (multiplication) kernel of matrix A

Kernel methods

$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric, nonnegative definite kernel

Write the eigenvalue expansion of k as

$$k(X, X') = \sum_{j=1}^{\infty} \theta_j \phi_j(X) \phi_j(X')$$

with

- $\theta_j \geq 0$ (nonnegative definite)
- $\left|\left|(\theta_j)_{j=1}^{\infty}\right|\right|_2 = \sum_{j=1}^{\infty} \theta_j^2 < \infty$
- The ϕ_j are orthogonal eigenfunctions: $\int \phi_j \phi_{j'} = \delta_{j,j'}$

(This is called Mercer's theorem, and such a k is called a Mercer kernel)

Kernel: Example

Back to polynomial terms/interactions:

Form

$$k_d(X, X') = (X^\top X' + 1)^d$$

k_d has $M = \binom{p+d}{d}$ eigenfunctions

These span the space of polynomials in $\mathbb{R}^p$ with degree d

Kernel: Example

Example: Let $d = p = 2 \Rightarrow M = 6$ and

$$\begin{aligned} k(u,v) &= 1 + 2u_1v_1 + 2u_2v_2 + u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2 \\ &= \sum_{k=1}^{M} \Phi_k(u)\Phi_k(v) \\ &= \Phi(u)^\top \Phi(v) \\ &= \langle \Phi(u), \Phi(v) \rangle \end{aligned}$$

where

$$\Phi(v)^\top = (1, \sqrt{2}v_1, \sqrt{2}v_2, v_1^2, v_2^2, \sqrt{2}v_1v_2)$$

Important: These equalities are everything that makes kernelization work!

Kernel: Conclusion

Let's recap:

$$\begin{aligned} k(u,v) &= 1 + 2u_1v_1 + 2u_2v_2 + u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2 \\ &= \langle \Phi(u), \Phi(v) \rangle \end{aligned}$$

- Some methods only involve features via inner products $X^\top X' = \langle X, X' \rangle$

  (We've explicitly seen two: ridge regression and support vector classifiers)
- If we make transformations of X to $\Phi(X)$, the procedure depends on $\Phi(X)^\top \Phi(X') = \langle \Phi(X), \Phi(X') \rangle$
- Crucial: We can compute this inner product via the kernel:

$$k(X, X') = \langle \Phi(X), \Phi(X') \rangle$$

Kernel: Conclusion

Instead of creating a very high dimensional object via transformations, choose a kernel k

Now, the only thing left to do is form the outer product of kernel evaluations

$$\mathbb{K} = [k(X_i, X_{i'})]_{1 \leq i, i' \leq n}$$

```
x = c(1,2,3)# n = 3
k = function(x,y){ return(x + y + x*y)}
> outer(x,x,k)
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    5    8   11
[3,]    7   11   15
```

(Kernel) SVMs

Kernel SVM

Recall:

$$\frac{1}{2}\,||\beta||_2^2 - \sum_{i=1}^{n} \alpha_i [Y_i(X_i^\top \beta + \beta_0) - 1]$$

Derivatives with respect to β and β_0 imply:

- $\beta = \sum_{i=1}^{n} \alpha_i Y_i X_i$
- $0 = \sum_{i=1}^{n} \alpha_i Y_i$

Write the solution function

$$h(X) = \beta_0 + \beta^\top X = \beta_0 + \sum_{i=1}^{n} \alpha_i Y_i X_i^\top X$$

Kernelize the support vector classifier $\Rightarrow$ support vector machine (SVM):

$$h(X) = \beta_0 + \sum_{i=1}^{n} \alpha_i Y_i k(X_i, X)$$

General kernel machines

After specifying a kernel function, it can be shown that many procedures have a solution of the form

$$\hat{f}(X) = \sum_{i=1}^{n} \gamma_i k(X, X_i)$$

For some $\gamma_1, \ldots, \gamma_n$

Also, this is equivalent to performing the method in the space given by the eigenfunctions of k

$$k(u, v) = \sum_{j=1}^{\infty} \theta_j \phi_j(u) \phi_j(v)$$

Also, (the) feature map is

$$\Phi = [\phi_1, \ldots, \phi_p, \ldots]$$

Kernel SVMs

Hence (and luckily) specifying Φ itself unnecessary,

(Luckily, as many kernels have difficult to compute eigenfunctions)

We need only define the kernel that is symmetric, positive definite

Some common choices for SVMs:

- Polynomial: $k(x, y) = (1 + x^\top y)^d$
- Radial basis: $k(x, y) = e^{-\tau ||x-y||_b^b}$

  (For example, $b = 2$ and $\tau = 1/(2\sigma^2)$ is (proportional to) the Gaussian density)

Kernel SVMs: Summary

Reminder: the solution form for SVM is

$$\beta = \sum_{i=1}^{n} \alpha_i Y_i X_i$$

Kernelized, this is

$$\beta = \sum_{i=1}^{n} \alpha_i Y_i \Phi(X_i)$$

Therefore, the induced hyperplane is:

$$h(X) = \Phi(X)^\top \beta + \beta_0 = \sum_{i=1}^{n} \alpha_i Y_i \langle \Phi(X), \Phi(X_i) \rangle + \beta_0$$

$$= \sum_{i=1}^{n} \alpha_i Y_i k(X, X_i) + \beta_0$$

The final classification is still $\hat{g}(X) = \mathrm{sgn}(\hat{h}(X))$

SVMs via penalization

SVMs via penalization

NOTE: SVMs can be derived from penalized loss methods

The support vector classifier optimization problem:

$$\min_{\beta_0,\beta} \frac{1}{2}\,||\beta||_2^2 + \lambda \sum \xi_i \text{ subject to}$$

$$Y_i h(X_i) \geq 1 - \xi_i, \xi_i \geq 0,, \text{ for each } i$$

Writing $h(X) = \Phi(X)^\top \beta + \beta_0$, consider

$$\min_{\beta,\beta_0} \sum_{i=1}^{n} [1 - Y_i h(X_i)]_+ + \tau\,||\beta||_2^2$$

These optimization problems are the same!

(With the relation: $2\lambda = 1/\tau$)

SVMs via penalization

The loss part is the hinge loss function

$$\ell(X, Y) = [1 - Yh(X)]_+$$

The hinge loss approximates the zero-one loss function underlying classification

It has one major advantage, however: convexity

Surrogate losses: convex relaxation

Looking at

$$\min_{\beta,\beta_0} \sum_{i=1}^{n} [1 - Y_i h(X_i)]_+ + \tau \, ||\beta||_2^2$$

It is tempting to minimize (analogous to linear regression)

$$\sum_{i=1}^{n} \mathbf{1}(Y_i \neq \hat{g}(X_i)) + \tau \, ||\beta||_2^2$$

However, this is nonconvex (in $u = h(X)Y$)

A common trick is to approximate the nonconvex objective with a convex one

(This is known as convex relaxation with a surrogate loss function)

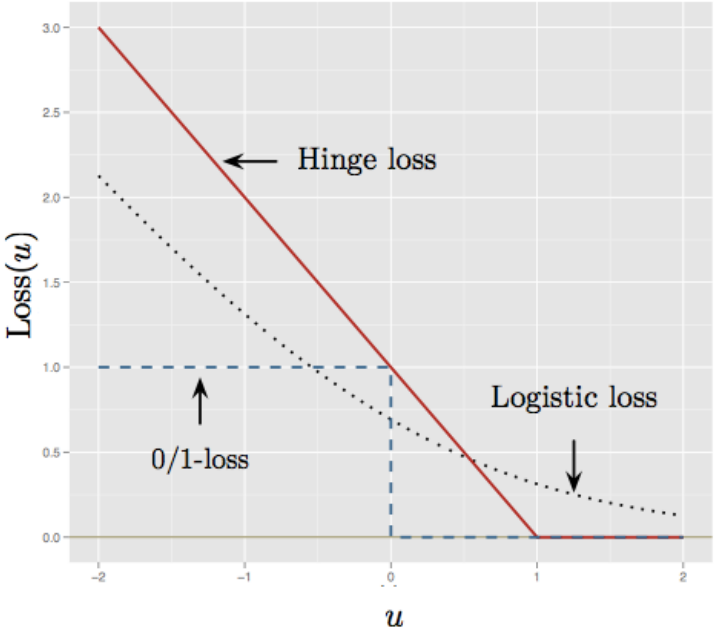
Surrogate losses

Idea: We can use a surrogate loss that mimics this function while still being convex

It turns out we have already done that! (twice)

- Hinge: $[1 - Yh(X)]_+$
- Logistic: $\log(1 + e^{-Yh(X)})$

Surrogate losses



SVMs in practice

GENERAL FUNCTIONS: The basic SVM functions are in the C++ library libsvm

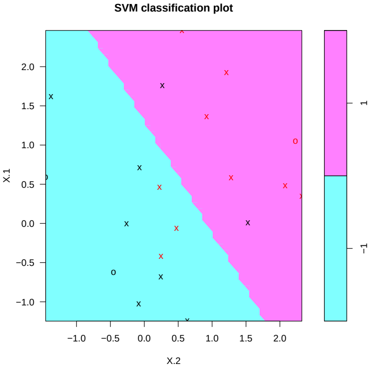
R PACKAGE: The R package e1071 calls libsvm

PATH ALGORITHM: `svmpath`

For a nice comparison of these approaches, see "Support vector machines in R"
(http://www.jstatsoft.org/v15/i09/paper)

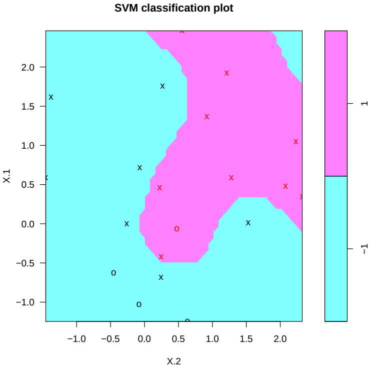
SVM EXAMPLE

```
tune.out = tune(svm,Y~.,data=dat,kernel="linear",
        ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
```



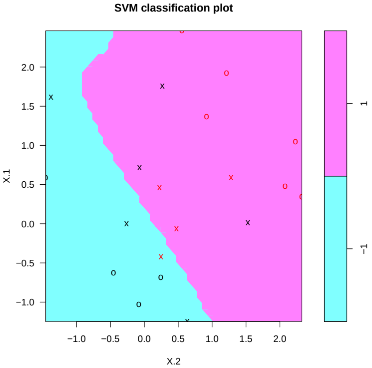
SVM EXAMPLE

```
tune.out = tune(svm,Y~.,data=dat,kernel="radial",
        gamma=c(1,2),
        ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
```

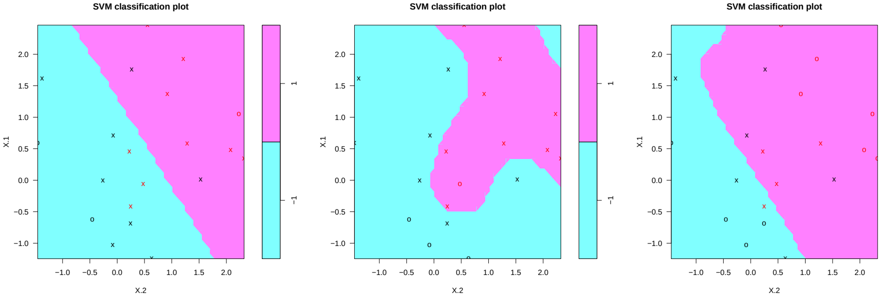


SVM example

```
tune.out = tune(svm,Y~.,data=dat,kernel="polynomial",
        degree=c(3,5,10),
        ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
```



SVM EXAMPLE



SVM classification plot
SVM classification plot
SVM classification plot
X.1
X.2

Multiclass classification

Multiclass SVMs

Sometimes, it becomes necessary to do multiclass classification

There are two main approaches:

- One-versus-one
- One-vesus-all

Multiclass SVMs: One-versus-one

Here, for G possible classes, we run $G(G-1)/2$ possible pairwise classifications

For a given test point X, we find $\hat{g}_k(X)$ for $k = 1, \ldots, G(G-1)/2$ fits

The result is a vector $\hat{G} \in \mathbb{R}^G$ with the total number of times X was assigned to each class

We report $\hat{g}(X) = \arg\max_g \hat{G}$

This approach uses all the class information, but can be slow

Multiclass SVMs: One-vesus-all

Here, we fit only G SVMs by respectively collapsing over all size $G - 1$ subsets of $\{1, \ldots, G\}$

(This is compared with $G(G-1)/2$ comparisons for one-versus-one) Take all

$\hat{h}_g(X)$ for $g = 1, \ldots, G$, where class g is coded 1 and "the rest" is coded -1

Assign $\hat{g}(X) = \arg\max_g \hat{h}_g(X)$

(Note that these strategies can be applied to any classifier)