

SUPPORT VECTOR MACHINES 2

-STATISTICAL LEARNING AND DATA MINING-

Lecturer: Darren Homrighausen, PhD

Preamble:

- We will generalize the Support Vector Classifier into the Support Vector Machine by leveraging “kernelization”
- Kernelization is the idea that we can replace inner products of observations ($\langle X_i, X_{i'} \rangle = X_i^\top X_{i'}$) with kernel evaluations ($k(X_i, X_{i'})$)
- This allows us to do dimension **expansion** without increasing the computational burden
- There is still a danger of overfitting with kernel methods, so we must regularize. Hence, we show that SVMs can be written as a penalized loss method, just like the logistic elastic net

OPTIMAL SEPARATING HYPERPLANE

REMINDER: The **optimal separating hyperplane** is produced by maximizing

$$\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^n \alpha_i \alpha_k Y_i Y_k \mathbf{x}_i^\top \mathbf{x}_k$$

subject to $\alpha_i \geq 0$

A similar result holds after the introduction of slack variables (e.g. **support vector classifiers**). In fact, the only difference is $\alpha_i \leq \lambda$ for each i)

IMPORTANT: The features only enter via

$$\mathbf{x}^\top \mathbf{x}' = \langle \mathbf{x}, \mathbf{x}' \rangle$$

(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

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

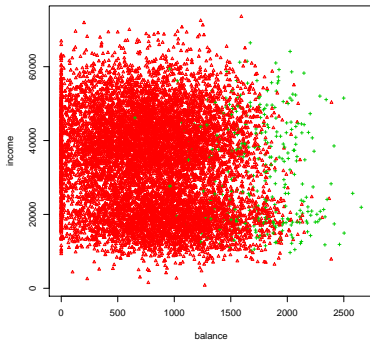
Again, we have the features enter only as

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

LOGISTIC REGRESSION: TRANSFORMATIONS

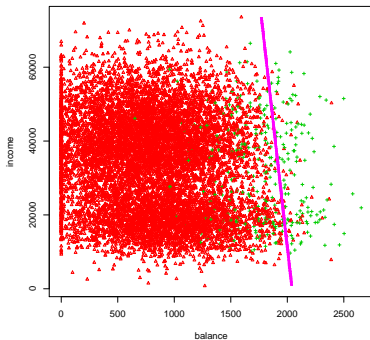
Let's look at the **default** data in ISL

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



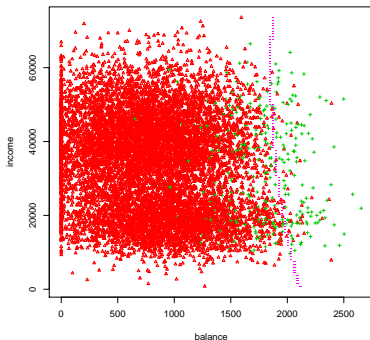
LOGISTIC REGRESSION: TRANSFORMATIONS

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



LOGISTIC REGRESSION: TRANSFORMATIONS

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



CONCLUSION: Linear rules in a transformed space can have nonlinear decisions in 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:

$$\Phi(X) = (\phi_1(X), \phi_2(X), \phi_3(X)) = (x_1, x_2, x_2^2)$$

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

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

This is **nonlinear** in the original x_1, x_2 feature space, but is linear in terms of $\Phi(X)$!

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
- We need to **regularize** to prevent overfitting

Can we form them in an automated fashion?

Kernel Methods

KERNEL: EXAMPLE

Back to polynomial terms/interactions $\Phi(X) = (x_1, x_2, x_2^2)$:

What if instead we could form a (kernel) function that produces these polynomial terms **automatically**?

WE CAN!

$$\rightarrow \text{Form } k(X, X') = (X^\top X' + 1)^d$$

This kernel **implicitly** forms all polynomials/interactions up to degree d

KERNEL: EXAMPLE

EXAMPLE: Let $d = p = 2$ and $u, v \in R^2$ be two vectors

$$\begin{aligned}k(u, v) &= 1 + 2u_1v_1 + 2u_2v_2 + u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2 \\&= \sum_{k=1}^M \Phi_k(u)\Phi_k(v) \quad (M = 6) \\&= \Phi(u)^\top \Phi(v) \\&= \langle \Phi(u), \Phi(v) \rangle\end{aligned}$$

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:

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

where

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

- 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)
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

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

Kernelize the SVC \Rightarrow support vector machine (SVM):

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

GENERAL KERNEL MACHINES

We can write the eigenvalue expansion of k as

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

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

Replacing inner products with kernel evaluations is equivalent to performing the unkernelized method in the space given by the **eigenfunctions** of k with **feature map** $\Phi = [\phi_1, \dots, \phi_p, \dots]$

POLYNOMIAL EXAMPLE:

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

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

KERNEL SVMs

Hence (and luckily) specifying Φ itself unnecessary,

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

Some common choices for SVMs:

- **POLYNOMIAL:** $k(X, X') = (1 + X^\top X')^d$
- **RADIAL BASIS:** $k(X, X') = e^{-\tau \|X - X'\|_b^b}$

(For example, $b = 2$ and $\tau = \sigma^{-2}/2$ 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:

$$\begin{aligned} f(X) &= \Phi(X)^\top \beta + \beta_0 = \sum_{i=1}^n \alpha_i Y_i \langle \Phi(X), \Phi(X_i) \rangle + \beta_0 \\ &= \sum_{i=1}^n \alpha_i Y_i k(X, X_i) + \beta_0 \end{aligned}$$

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

SVMs via penalization

SVMs VIA PENALIZATION

NOTE: SVMs can be derived from **penalized loss** methods

The support vector classifier optimization problem:

$$\min_{\beta_0, \beta, \xi} \frac{1}{2} \|\beta\|_2^2 + \lambda \sum \xi_i \quad \text{subject to}$$
$$Y_i f(X_i) \geq 1 - \xi_i, \xi_i \geq 0, \text{ for each } i$$

Consider the alternative optimization problem:

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

These optimization problems are the same!

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

SVMs VIA PENALIZATION

The **loss** part is the **hinge loss function**

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

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

It has one major advantage, however: **convexity**

SURROGATE LOSSES

Looking at

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

It is tempting to minimize (analogous to linear regression)

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

However, this is **nonconvex**

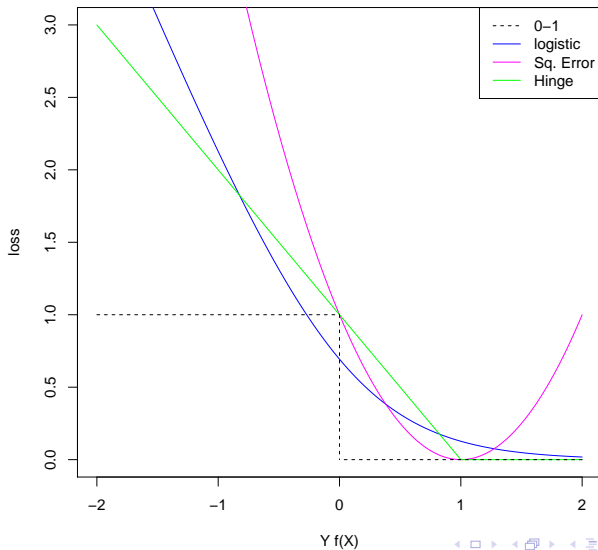
SURROGATE LOSSES

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

It turns out we have already done that! (two times)

- **HINGE:** $[1 - Yf(X)]_+$
- **LOGISTIC:** $\log(1 + e^{-Yf(X)})$

COMPARING LOSS FUNCTIONS



SVMs IN PRACTICE

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

R PACKAGE: The **R** package **e1071** calls **libsvm**

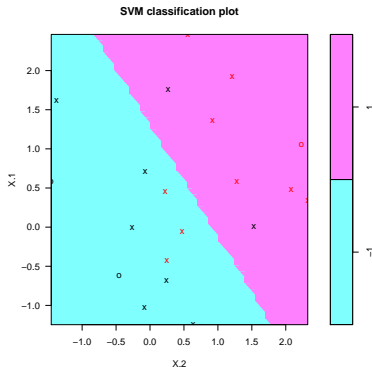
PATH ALGORITHM: **svmpath**

For a nice comparison of these approaches, see “Support vector machines in **R**”

(<http://www.jstatsoft.org/v15/i09/paper>)

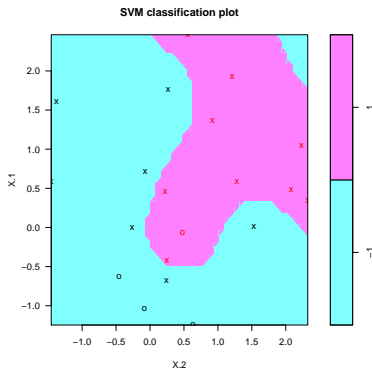
SVM EXAMPLE

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



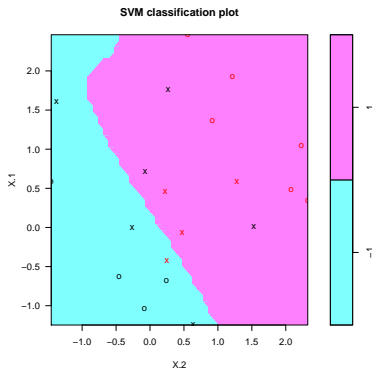
SVM EXAMPLE

```
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)))
```

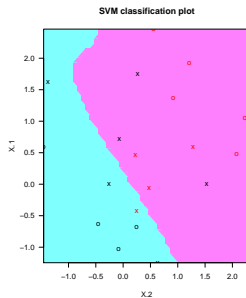
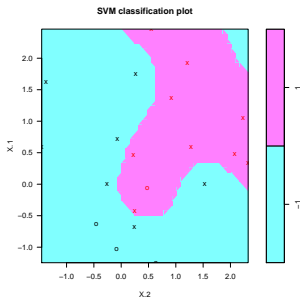
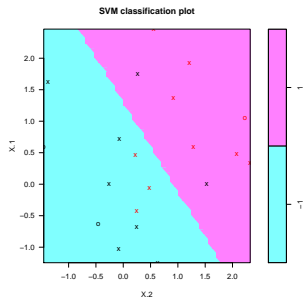


SVM EXAMPLE

```
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



Multiclass SVMs

MULTICLASS SVMs

Sometimes, it becomes necessary to do multiclass classification

There are two main approaches:

- One-versus-one
- One-versus-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 $\hat{f}_k(X)$ for $k = 1, \dots, G(G - 1)/2$ 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 $\hat{f}(X) = \arg \max_g \hat{G}$

This approach leverages all the class information, but can be **REALLY** slow

(It does have the advantage of only using at any one time the training observations i such that Y_i has either of two classes we are considering)

MULTICLASS SVMs: ONE-VERSUS-ALL

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

Take all $\hat{h}_g(X)$ for $g = 1, \dots, G$, where class g is coded 1 and “the rest” is coded -1

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

Postamble:

- We will generalize the Support Vector Classifier into the Support Vector Machine by leveraging “kernelization”

(Reminder: we saw this before in ridge regression)

- Kernelization is the idea that we can replace inner products of observations ($\langle X_i, X_{i'} \rangle = X_i^\top X_{i'}$) with kernel evaluations ($k(X_i, X_{i'})$)

(This can be seen in the form of the solution

$$\hat{f}(X) = \sum_{i=1}^n \alpha_i Y_i X^\top X_i \rightarrow \sum_{i=1}^n \alpha_i Y_i k(X, X_i))$$

- This allows us to do dimension **expansion** without increasing the computational burden
- There is still a danger of overfitting with kernel methods, so we must regularize. Hence, we show that SVMs can be written as a penalized loss method, just like the logistic elastic net

(This is the “hinge loss function”)