Principles of Statistical Machine Learning
Introduction to Support Vector Machine Classification

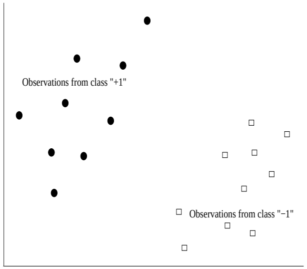
Ernest Fokoué

School of Mathematical Sciences
Rochester Institute of Technology
Rochester, New York, USA

Principles of Statistical Machine Learning
STAT 747-Autumn Semester 2017

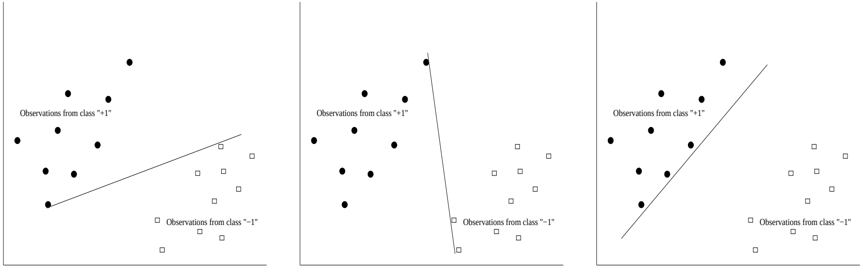
Introduction to Support Vector Machines Classification

Let $\{(\mathbf{x}_i, \mathrm{y}_i),\ i = 1, \cdots, n\}$ be a random sample of pairs where $\mathbf{x}_i \in \mathbb{R}^p$ is the vector of explanatory variables, and $\mathrm{y}_i \in \{-1, +1\}$ is the binary response. The support vector machine method of classification seeks to use the data to estimate the decision boundary that best separates the two classes. When the two populations are linearly separable, the decision boundary between the two classes is a hyperplane. Below is a simple example where the dimension of the vector to be classified is $p = 2$.



Support Vector Machines Decision Boundaries

- *The above problem is linearly separable, and we can visually explore some possible classifiers. Each of the following decision boundary is linear and seems to do well.*

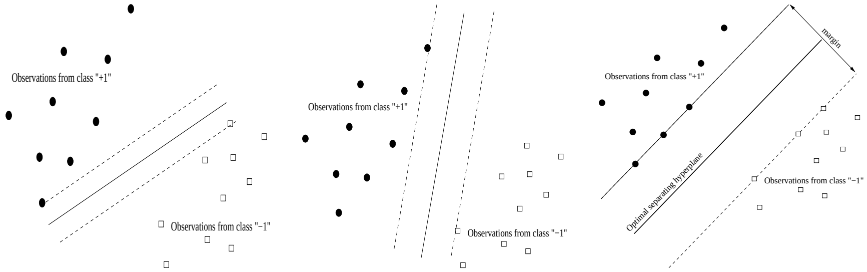


- *The decision boundary in this case is the hyperplane*

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0. \tag{1}$$

Support Vector Machines and Large Margins

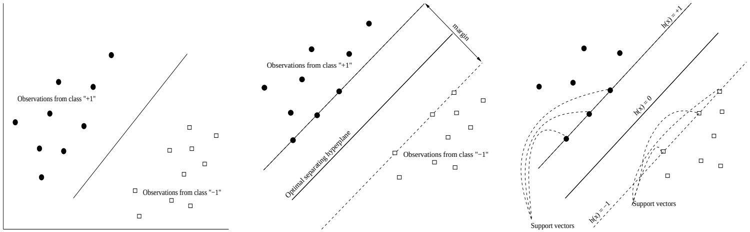
- *SVM does not just seek any decision boundary. It seeks one that produces the largest margin. That's why SVM classifiers belong to the so-called Large Margin Classifiers family. In other words, although all the above decision boundaries are reasonable, some are better than others according to the large margin principle.*


Observations from class "+1"
Observations from class "−1"
Observations from class "+1"
Observations from class "−1"
Observations from class "+1"
margin
Optimal separating hyperplane
Observations from class "−1"

- *The plot on the right shows the largest margin for what is clearly the best of the decision boundaries considered.*

Support Vector Machines and Large Margins

- *Putting all the pieces of the above reasoning together, our best decision boundary according to the large margin principle can be described in the following plots*


Observations from class "+1"
Observations from class "−1"
Observations from class "+1"
margin
Optimal separating hyperplane
Observations from class "−1"
h(x) = +1
h(x) = 0
h(x) = −1
Support vectors
Support vectors

- *Using analytic geometry, the margin is found to be*

$$\rho = \frac{2}{\|\boldsymbol{w}\|} \tag{2}$$

- *The goal in SVM classification is to build the classifier that maximizes the margin.*

Support Vector Machines Formulation

For the above linearly separable problem, the support vector machine classifier $\hat{f}$ *is obtained by solving a constrained minimization problem with objective function*

$$E(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2$$

subject to

$$\mathrm{y}_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \quad i \in [n]$$

Specifically, we have

$$\hat{f}(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{j=1}^{|\mathbf{s}|} \hat{\alpha}_{s_j} \mathrm{y}_{s_j} \boldsymbol{x}_{s_j}^\top \boldsymbol{x} + \hat{b}\right) \tag{3}$$

where $s_j \in \{1, 2, \cdots, n\}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_{|\mathbf{s}|}\}$ *and* $|\mathbf{s}| \lll n$. *The vectors*

$$\boldsymbol{x}_{s_1}, \boldsymbol{x}_{s_2}, \cdots, \boldsymbol{x}_{s_{|\mathbf{s}|}}^\top$$

are special and a referred to as support vectors - hence the name support vector machines (SVM).

Support Vector Machines Formulation

- *Clearly, from the formula of* (3)*, the support vectors are the only ones needed for predicting responses for new vectors. The SVM classifier is said to achieve a sparse representation in data space, because* $|\mathrm{s}| \lll n$.
- *This is the case because, in its raw form,* $\hat{f}$ *can be written in terms of all the* n *points in the sample, namely,*

$$\hat{f}(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{j=1}^{n} \hat{\alpha}_j \mathrm{y}_j \boldsymbol{x}_j^\top \boldsymbol{x} + \hat{b}\right).$$

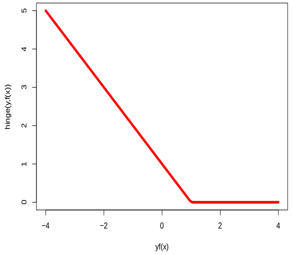
- *However, since it is the case the* $\alpha_j = 0$ *for all non support vectors, it makes sense to only write the estimator in terms of its non zero coefficients. Hence, the sparsity.*

Note: In practice, many interesting problems are not linearly separable. We may either have most of the data well separable linearly, but with a sizeable number of observations falling on the wrong side of the decision boundary, or we may be in the presence of a complete nonlinear case.

Support Vector Machines and the Hinge Loss

The hinge loss is defined as

$$\ell(\mathrm{y}, f(\boldsymbol{x})) = (1-\mathrm{y}f(\boldsymbol{x}))_+ = \max(0, 1-\mathrm{y}f(\boldsymbol{x})) = \begin{cases} 0 & \mathrm{y} = f(\boldsymbol{x}) \\ 1 - \mathrm{y}f(\boldsymbol{x}) & \mathrm{y} \neq f(\boldsymbol{x}) \end{cases}$$



Notice that the hinge loss is simply a functional formulation of the constraint in the original SVM paradigm.

Support Vector Machines and the Hinge Loss

- *With the hinge loss, the Support Vector Machine classifier can be formulated as*

$$\texttt{Minimize } E(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(1 - \mathrm{y}_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))_+$$

`subject to`

$$\|\boldsymbol{w}\|_2^2 < \tau.$$

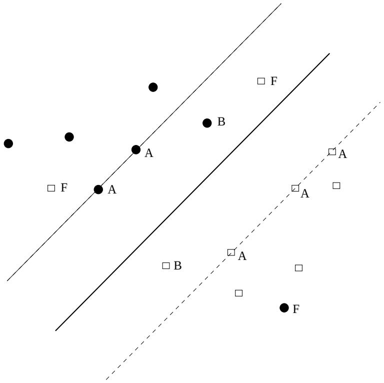
- *Which is equivalent in regularized (lagrangian) form to*

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^q}\left\{\frac{1}{n}\sum_{i=1}^{n}(1 - \mathrm{y}_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b))_+ + \lambda\|\boldsymbol{w}\|_2^2\right\}$$

- *It is important in this formulation to maintain the responses coded as* $\{-1, +1\}$.

Support Vector Machines For Non Linearly Separable

- *Consider the following display, and notice that how ever hard you try to find a large margin linear classifier that separable all the data into two distinct classes, you will have a point that falls on the wrong side.*



- *Exercise:* *Describe the meaning of the labels/codes* `A`, `B`, *and* `F`.

Support Vector Machines For Non Linearly Separable

- *It is clear here that no matter what, there will be errors. The simple way around it is to define a characteristic for errors and modify our objective function in such a way that the total number of errors is minimized as par of building the best separating hyperplane.*
- *For the so-called C-SVC or C-Support Vector Classification, the support vector machine classifier $\hat{f}$ is obtained by solving a constrained minimization problem with objective function*

$$E(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i.$$

subject to

$$\mathrm{y}_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i \in [n],$$

where $\xi_i \geq 0, \quad i \in [n]$ and C is a constant to be specified by the experimenter.

$$\texttt{Number of Errors} = \sum_{i=1}^{n} I(\xi_i > 1)$$

Support Vector Machines For Non Linearly Separable

- *For the so-called ν-SVC or nu-Support Vector Classification, the support vector machine classifier $\hat{f}$ is obtained by solving a constrained maximization problem with objective function*

$$E(\boldsymbol{w}, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \rho\nu + \frac{1}{n}\sum_{i=1}^{n}\xi_i.$$

subject to

$$\mathrm{y}_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq \rho - \xi_i,$$

where $\xi_i \geq 0$ and $\rho \geq 0$. where ρ and ν are parameters to be estimated from the data. As a result, for each $\boldsymbol{x}$, the estimated SVM response is

$$\hat{f}(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{j=1}^{|\mathbf{s}|}\hat{\alpha}_{s_j}\mathrm{y}_{s_j}\boldsymbol{x}_{s_j}^\top\boldsymbol{x} + \hat{b}\right)$$

where $s_j \in \{1, 2, \cdots, n\}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_{|\mathbf{s}|}\}$ and $|\mathbf{s}| \lll n$.

SVM for Nonlinear Decision Boundaries

When the decision boundary is nonlinear, the problem gets understandably more complicated and requires more sophisticated tools. Below is an extreme case:

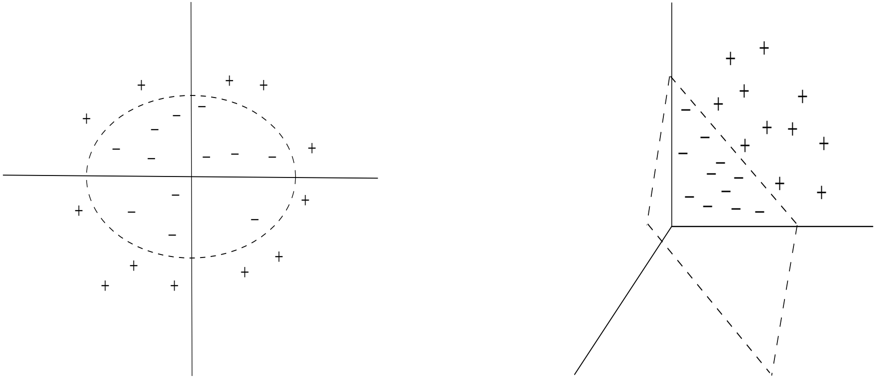


Figure: *(Left) In two dimensions, the problem is heavily nonlinear. (right) Pushed up (projected) to three dimensions, the very same problem is linearly separable.*

SVM for Nonlinear Decision Boundaries

- *When the decision boundary is nonlinear, the support vector machine classifier $\hat{f}$ is now obtained by solving a constrained minimization problem with objective function*

$$E(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i.$$

  subject to

$$\mathrm{y}_i(\boldsymbol{w}^\top \Phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i,$$

  where $\xi_i \geq 0$ and C is a constant to be specified by the experimenter, and $\Phi(\boldsymbol{x}_i)$ is the projection of $\boldsymbol{x}_i$ into the feature space $\mathcal{F}$. The solution $\boldsymbol{w}$ is of the form

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \mathrm{y}_i \Phi(\boldsymbol{x}_i)$$

  Inherently, $\alpha_i \neq 0$, when the constraint is satisfied at $(\boldsymbol{x}_i, ry_i)$.
- *The α_i are determined by solving a quadratic programming problem.*

SVM for Nonlinear Decision Boundaries

Basically, for each point $\boldsymbol{x}_i$ *in* q*-dimensional space, one finds the projected equivalent* $\Phi(\boldsymbol{x}_i)$ *in the in the feature space* $\mathcal{F}$*. As a result, we now have*

$$\hat{f}(\boldsymbol{x}) = \text{sign}\left(\sum_{j=1}^{|\mathbf{s}|} \hat{\alpha}_{s_j} \mathrm{y}_{s_j} \Phi(\boldsymbol{x}_{s_j})^\top \Phi(\boldsymbol{x}) + \hat{b}\right)$$

It turns out that instead of having to project using $\Phi(\cdot)$*, one can directly use a special function known as kernel. In other words, in most cases, one can find a kernel such that*

$$K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) = \langle \Phi(\boldsymbol{x}_{s_j}), \Phi(\boldsymbol{x}) \rangle = \Phi(\boldsymbol{x}_{s_j})^\top \Phi(\boldsymbol{x}).$$

This is known as the kernel trick*. As a result of the kernelization, the SVM classifier delivers for each* $\boldsymbol{x}$*, the estimated response*

$$\hat{f}(\boldsymbol{x}) = \text{sign}\left(\sum_{j=1}^{|\mathbf{s}|} \hat{\alpha}_{s_j} \mathrm{y}_{s_j} K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) + \hat{b}\right)$$

where $s_j \in \{1, 2, \cdots, n\}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_{|\mathbf{s}|}\}$ *and* $|\mathbf{s}| \lll n$.

Support Vector Machines and Kernels

- *As a result of the kernelization, the SVM classifier delivers for each $\boldsymbol{x}$, the estimated response*

$$\hat{f}(\boldsymbol{x}) = \text{sign}\left(\sum_{j=1}^{|\mathbf{s}|} \hat{\alpha}_{s_j} \mathrm{y}_{s_j} K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) + \hat{b}\right)$$

  where $s_j \in \{1, 2, \cdots, n\}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_{|\mathbf{s}|}\}$ *and* $|\mathbf{s}| \lll n$.
- *The linearly separable case above corresponds to the basic Euclidean space inner product, namely*

$$K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) = \boldsymbol{x}_{s_j}^{\top} \boldsymbol{x}.$$

  which is referred to as the linear kernel or vanilla kernel.
- *Many sophisticated kernels are commonly used to tackle challenging machine learning and data mining tasks. Some of those kernels are given in the following pages.*

SVM Learning via Quadratic Programming

- *When the decision boundary is nonlinear, the α_i's in the expression of the support vector machine classifier $\hat{f}$ are determined by solving the following quadratic programming problem*

$$\texttt{Maximize } E(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j \mathrm{y}_i\mathrm{y}_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

`subject to`

$$0 \leq \alpha_i \leq C \ (i = 1, \cdots, n) \quad \texttt{and} \quad \sum_{i=1}^{n} \alpha_i \mathrm{y}_i = 0.$$

- *The above formulation is an instance of the general QP*

$$\texttt{Maximize}\left\{ -\frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha} + \mathbf{1}^\top \boldsymbol{\alpha} \right\}$$

`subject to`

$$\boldsymbol{\alpha}^\top \mathbf{y} = 0 \quad \textit{and} \quad \alpha_i \in [0, C], \quad \forall i \in [n].$$

SVM Learning via Quadratic Programming in R

- *The quadratic programming problem*

$$\texttt{Maximize}\left\{-\frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha} + \mathbf{1}^\top\boldsymbol{\alpha}\right\}$$

subject to $\boldsymbol{\alpha}^\top\mathbf{y} = 0$ *and* $\alpha_i \in [0, C],\ \forall i \in [n]$. *is equivalent to*

$$\texttt{Minimize}\left\{\frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha} - \mathbf{1}^\top\boldsymbol{\alpha}\right\}$$

subject to $\boldsymbol{\alpha}^\top\mathbf{y} = 0$ *and* $\alpha_i \in [0, C], \quad \forall i \in [n]$.

- *Which is solved with the R package kernlab via the function* ipop()

$$\texttt{Minimize}\left\{\boldsymbol{c}^\top\boldsymbol{\alpha} + \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{H}\boldsymbol{\alpha}\right\}$$

subject to $b \leq \boldsymbol{A}\boldsymbol{\alpha} \leq b + r$ *and* $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$.

Estimation of b

Support Vector Machines and Kernels

- *As a result of the kernelization, the SVM classifier delivers for each $\boldsymbol{x}$, the estimated response*

$$\hat{f}(\boldsymbol{x}) = \text{sign}\left(\sum_{j=1}^{|\mathbf{s}|} \hat{\alpha}_{s_j} \mathrm{y}_{s_j} K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) + \hat{b}\right)$$

  where $s_j \in \{1, 2, \cdots, n\}$, $\mathbf{s} = \{s_1, s_2, \cdots, s_{|\mathbf{s}|}\}$ *and* $|\mathbf{s}| \lll n$.

- *The linearly separable case above corresponds to the basic Euclidean space inner product, namely*

$$K(\boldsymbol{x}_{s_j}, \boldsymbol{x}) = \boldsymbol{x}_{s_j}^\top \boldsymbol{x}.$$

  which is referred to as the linear kernel or vanilla kernel.

- *Many sophisticated kernels are commonly used to tackle challenging machine learning and data mining tasks. Some of those kernels are given in the following pages.*

SVM Learning via Quadratic Programming in R

```
n <- 100
x <- cbind(runif(n, -2.0, 2.0), runif(n, -2.0, 2.0))
y <- ifelse(x[,2] - linear.boundary(x[,1])> 0, 1, -1)
C <- 0.1              # Errors hyperparameter

lin <- vanilladot()
x   <- standardize(x)
H <- kernelPol(lin,x,,y)
c <- matrix(rep(-1,n))
A <- t(y)
b <- 0
l <- matrix(rep(0,n))
u <- matrix(rep(C,n))
r <- 0

sv <- ipop(c,H,A,b,l,u,r)   #   QP learning for SVM
```

SVM Classification with Linear Boundary

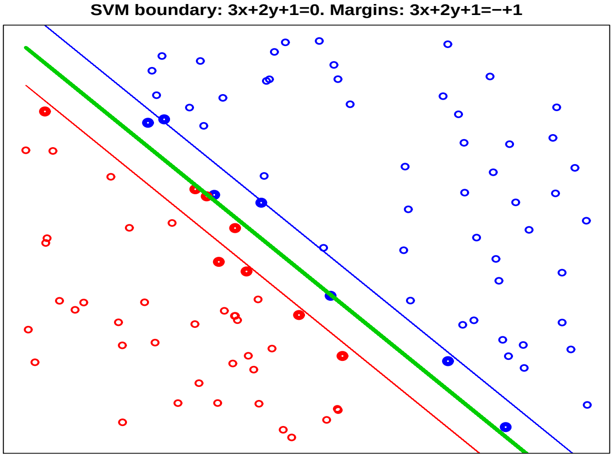


Figure: *Linear SVM classifier with a relatively small margin*

SVM Classification with Linear Boundary

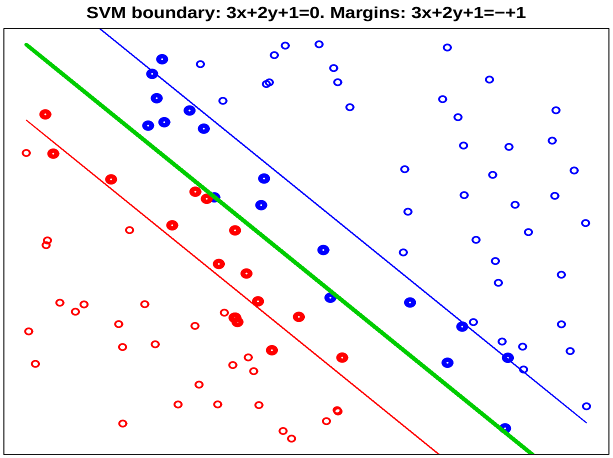


Figure: *Linear SVM classifier with a relatively large margin*

SVM Classification with Nonlinear Boundary

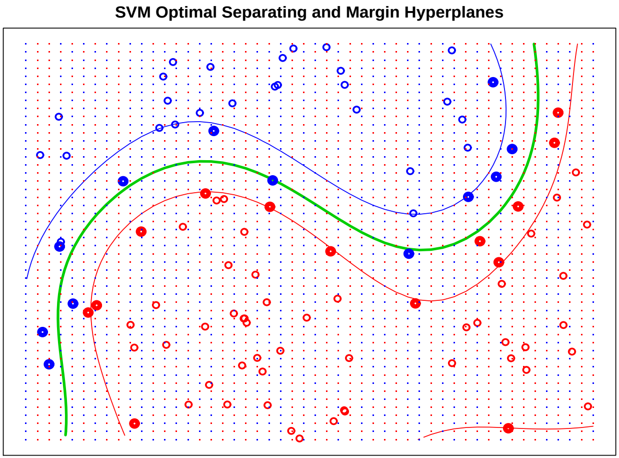


Figure: *Nonlinear SVM classifier with a relatively small margin*

Modeling with Kernels as Measures of Similarity

The most commonly used kernels include

- *Linear kernel (vanilla)*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \boldsymbol{x}_i^\top \boldsymbol{x}_j \tag{4}$$

- *Polynomial Kernel of degree d*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\beta \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \tau)^d = (\beta \boldsymbol{x}_i^\top \boldsymbol{x}_j + \tau)^d \tag{5}$$

  where β is the scale and τ is the offset, and d is the degree.

- *Gaussian Radial Basis Function (RBF) Kernel*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\delta \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2\right) \tag{6}$$

  where δ is the bandwidth.

- *Laplace Radial Basis Function (RBF) Kernel*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\delta \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_1\right)$$

  where δ is the bandwidth.

Modeling with Kernels as Measures of Similarity

The most commonly used kernels include

- *Hyperbolic tangent kernel*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh\left(\beta\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle + \tau\right) = \tanh\left(\beta \boldsymbol{x}_i^\top \boldsymbol{x}_j + \tau\right)$$

- *ANOVA radial basis kernel*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{\ell=1}^{q} \exp\left(-\delta(\mathbf{x}_{i\ell} - \mathbf{x}_{j\ell})^2\right)\right)^d$$

where δ is the bandwidth, and d is the degree. This is an isotropic version that can be extended to an anisotropic with a different δ_ℓ for each dimension.

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\sum_{\ell=1}^{q} \exp\left(-\delta_\ell(\mathbf{x}_{i\ell} - \mathbf{x}_{j\ell})^2\right)\right)^d$$

Modeling with Kernels as Measures of Similarity

The most commonly used kernels include

- *Spline kernel*

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{\ell=1}^{q} \mathcal{K}(\mathbf{x}_{i\ell}, \mathbf{x}_{j\ell})$$

where using $\mathbf{x}_{i\ell} \wedge \mathbf{x}_{j\ell} = \min(\mathbf{x}_{i\ell}, \mathbf{x}_{j\ell})$, *we have*

$$\mathcal{K}(\mathbf{x}_{i\ell}, \mathbf{x}_{j\ell}) = 1 + \mathbf{x}_{i\ell}\mathbf{x}_{j\ell}(\mathbf{x}_{i\ell} \wedge \mathbf{x}_{j\ell}) - \frac{\mathbf{x}_{i\ell} + \mathbf{x}_{j\ell}}{2}(\mathbf{x}_{i\ell} \wedge \mathbf{x}_{j\ell})^2 + \frac{1}{3}(\mathbf{x}_{i\ell} \wedge \mathbf{x}_{j\ell})^3$$

Once the kernel is chosen (defined), the Gram matrix can be generated

$$K = \begin{bmatrix} \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_1) & \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & \mathcal{K}(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix}. \tag{7}$$

Aspects of Kernels and their Applications

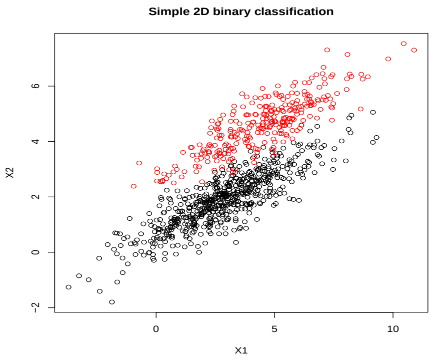
- *The linear (vanilla) kernel tends to work on very high dimensional data that are very sparse like the kind that arises in text mining and text classification (categorization)*
- *The polynomial kernel has been known to perform very well on image processing tasks*
- *The Spline and ANOVA kernels have been found to perform typically well on regression tasks*
- *The Gaussian RBF kernel and the Laplace RBF kernel seem to be the default kernels of choice when there is limited prior knowledge about the data. This is probably due to the fact they are more general and will help model arbitrarily complex problem just by changing the bandwidth.*
- *The hyperbolic tangent (sigmoid) kernel reminds us of artificial neural networks*

Aspects of Kernels and their Applications

- *For each of the above kernels, one or two tuning parameters (hyperparameters) need to be estimated. This estimation is usually part of any decent software package that implements the support vector machine paradigm. Cross Validation is also quite often used for such estimation.*
- *The procedures for estimating the parameters of the SVM machinery are usually borrowed from the traditional field of constrained optimization. The details are beyond the scope of this course.*
- *One of the main tools used by SVM is quadratic programming which is fortunately well implemented in R.*
  - *Indeed,* `quadprog`*() is an R function that does quadratic programming.*
  - *The package kernlab also a function* `iprop`*() that does quadratic programming.*
- *kernlab and e1071 are two R packages to provide SVM implementations.*

Simulated Example of Binary Classification

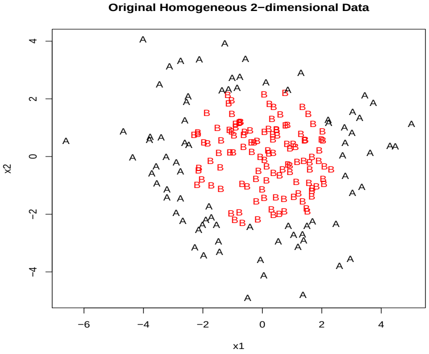
To gain deeper insights, consider the following cloud of points:



The logistic regression model of (**??**) *should do well on the above.*

Homogeneous Doughnut Classification Problem

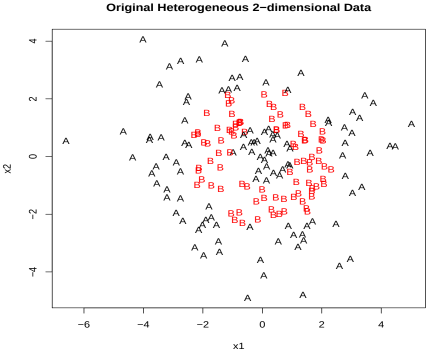
However, if we have the following cloud of points:



- *How well will logistic regression - under its underlying linearity assumption - fare on this data? Uhmmmm!*

Heterogeneous Doughnut Classification Problem

Better yet, if the cloud of points is:



- *How well will logistic regression - under its underlying linearity assumption - fare on this data? Pretty pitiful!*

Comparison of Performances of Classifiers

	Pattern recognition technique				
	SVM	QDA-Tr	QDA	Logit	Logit-Tr
Homogeneous	*0.98*	*0.98*	*0.95*	*0.52*	*1.00*
Heterogeneous	*0.95*	*0.83*	*0.75*	*0.54*	*0.77*

Table: Average accuracy of classifiers over 50 replications of the same task.

- *Failure of linear logistic*
- *Success with nonlinear mapping*

Question: How is the nonlinearity modeled so that such tremendous improvement is achieved?

Most Popular Kernels

- *The polynomial kernel of Vladimir Vapnik. (Professor Vladimir Vapnik is the co-inventor of the Support Vector Machine paradigm)*

  $$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = (b\mathbf{x}_i^\top \mathbf{x}_j + a)^d$$

  where d is the degree of the polynomial, b is the scale parameter and a is the offset parameter.

- *The Gaussian Radial Basis function kernel*

  $$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\omega^2}\right)$$

  This is arguably the most popular kernel because of its flexibility, but also because it is found be provide a decent approximation to many real life problems.

Most Popular Kernels

- *The Laplace kernel*

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|\right)$$

  This is the ℓ_1-norm counterpart of the RBF kernel. Although apparently similar, they sometimes produce drastically different results when applied to practical situations.

- *Hyperbolic tangent kernel*

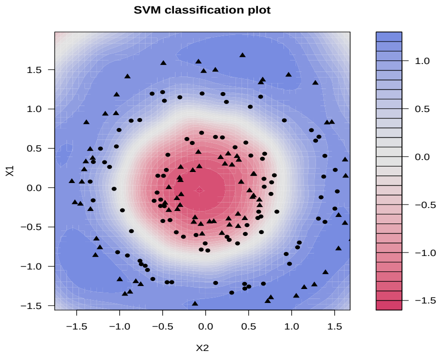
$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(b\mathbf{x}_i^\top \mathbf{x}_j + a\right)$$

- *Other kernels:(a) string kernels used in the text categorization (b) ANOVA kernels (c) Spline Kernels (d) Bessel kernels*

Note: A kernel in this context is essentially a measure of similarity.

SVM on Doughnut Data

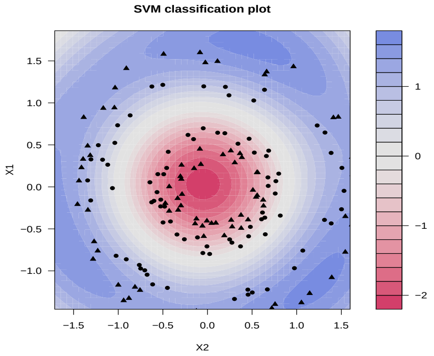
With the Laplace kernel, SVM delivers



- *Performance: Separation is clear, and accuracy is 98.25%, with 174 support vectors out of the $n = 400$ observations. Pretty good!*

SVM on Doughnut Data

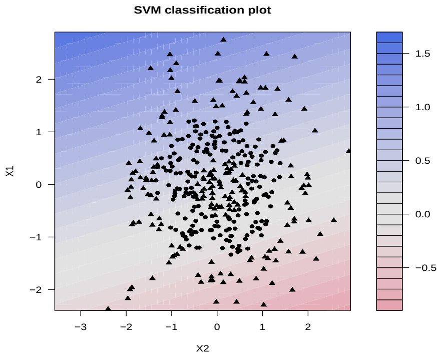
With the Gaussian Radial Basis Function kernel, SVM delivers



- *Performance: Separation is clear, and accuracy is* 96.5%*, with* 165 *support vectors out of the* $n = 400$ *observations. Pretty good!*

SVM on Doughnut Data

With the polynomial kernel, SVM delivers



- *Performance: Nothing going (confusion), and accuracy is 50.25%, with 387 support vectors out of the $n = 400$ observations. Pretty pitiful, total failure!*

Appeal of Kernels

Consider the Gaussian RBF kernel for illustration

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\omega^2}\right)$$

- *Modeling of Nonlinearity in Extremely High dimensional: The choice of ω corresponds to the selection of an entire class of function, which can be very large and very sophisticated.*
- *Interesting intuition: Similarity measure provides some insights into why it works*

  $$\lim_{\|\mathbf{x}_i - \mathbf{x}_j\| \to 0} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = 1$$

  and

  $$\lim_{\|\mathbf{x}_i - \mathbf{x}_j\| \to \infty} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = 0$$

  Therefore, we have local approximation in the right neighborhood, which is a good thing!

Immediate Challenges with Kernel Methods

- *The matrix that results from the computation of all such pairs is known as the Gram Matrix, and is an $n \times n$ matrix given by*

$$\mathbf{K} = \left[\begin{array}{cccc} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_1, \mathbf{x}_n) \\ \mathcal{K}(\mathbf{x}_2, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \ddots & \cdots & \vdots \\ \mathcal{K}(\mathbf{x}_n, \mathbf{x}_1) & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_2) & \cdots & \mathcal{K}(\mathbf{x}_n, \mathbf{x}_n) \end{array}\right].$$

  This matrix grows with n, and can be temperamental!

- *The underlying model of equation* (**??**), *namely*

$$\eta(\mathbf{x}_i; \mathrm{v}, \mathbf{w}) = \mathrm{v} + \mathrm{w}_1\mathcal{K}(\mathbf{x}_i, \mathbf{x}_1) + \mathrm{w}_2\mathcal{K}(\mathbf{x}_i, \mathbf{x}_2) + \cdots + \mathrm{w}_n\mathcal{K}(\mathbf{x}_i, \mathbf{x}_n)$$

  has a total of $2^n - 1$ submodels. If one has to search this model space, then one has to go about it very efficiently, or else!!!!

Classification of E. coli Promoter Gene Sequences (DNA)

Context Description: *Promoter gene sequences have a region where a protein must make contact and the helical DNA sequence must have a valid conformation so that the two pieces of the contact region spatially align. DNA bases are coded as follows:*

a *adenine* **c** *cytosine* **g** *guanine* **t** *thymine.*

Data Description: *DNA sequences of promoters and non-promoters:* $n = 106$ *observations and* $p = 58$ *variables.*

- *The first variable Class is the binary response variable coded as* $+$ *for a promoter gene and - for a non-promoter gene.*
- *The remaining* 57 *variables V2 to V58 are the explanatory variables describing the sequence.*

Research Question:

- *How does a biologist discriminate between a promoter and a non promoter gene sequence?*
- *How does one build a accurate classifier for this type of data?*

Classification of DNA Gene Sequences

The data: *An R package that explores this data is kernlab*

```
library(kernlab)
data(promotergene)
head(promotergene)
```

A quick look at the first 15 *variables of two of the observations*

```
+ gccttctccaaaac
- gaggtggctatgtg
```

Now let's look at all the 58 *variables of the above two observations*

```
+ gccttctccaaaacgtgtttttgttgttaattcggtgtagacttgtaaacctaaat
- gaggtggctatgtgtatgaccgaacgagtcaatcagaccgctttgactctggtatta
```

- *How did the biologist decide that the first was a promoter and the second a non promoter?*
- *How does one build a accurate classifier for this type of data?*

SVM Classification of DNA Gene Sequences

Support Vector Machine Classification

```
svm.gene <- ksvm(Class~.,data=promotergene,
                 kernel="laplacedot",
                 C=60,cross=4)
svm.est.class.gene <- predict(svm.gene,
                      promotergene[,-1])
table(promotergene$Class,svm.est.class.gene)
```

The confusion matrix in this case is given by

```
       +   -
    + 53   0
    -  0  53
```

- *The classification error is amazingly* 0*! Uhhmmm! [?]*
- *Will the test error be* 0 *if we split into training and test sets?*

SVM Classification of DNA Gene Sequences

SVM Classification: Now, let's use $2/3$ of the data for training the SVM and the remaining $1/3$ for testing. Training confusion matrix is

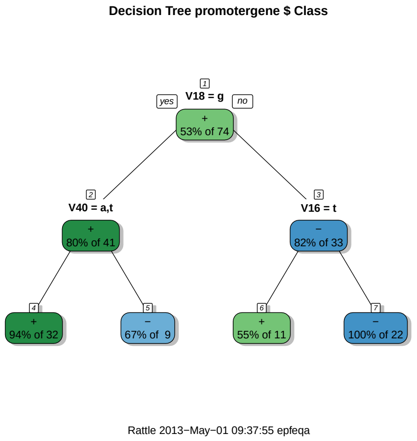
```
          svm.est.Y.tr
true.Y.tr  0  1
        0 35  0
        1  0 35
```

The Support Vector Machine Training Error is still 0. However, now we have the test confusion matrix

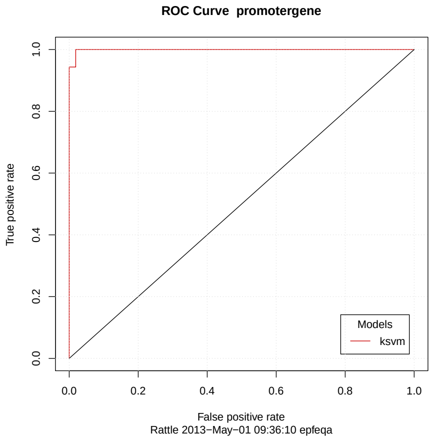
```
          svm.est.Y.te
true.Y.te  0  1
        0 16  2
        1  3 15
```

With that, the Support Vector Machine Test Error is 0.1388889.

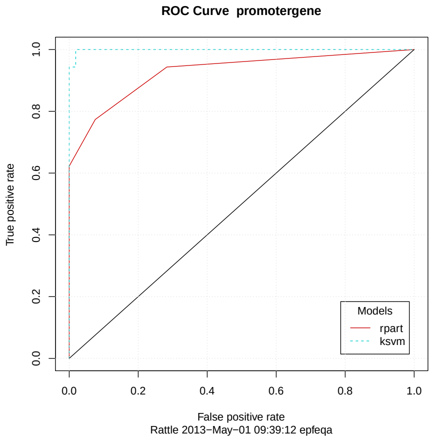
Plot of Tree Classification on Gene Data



ROC Curve of SVM on Gene Data



ROC Curve of SVM vs Tree on Gene Data



Examples of datasets with characteristics

	n	p	n/p	$\log(\lvert\texttt{cov}(\boldsymbol{X})\rvert)$	c
crabs	200	5	40.0	-10.04	2
pima	532	7	76.0	-1.60	2
spam	4601	57	80.7	-15.24	2
musk	476	166	2.9	-541.15	2
lymphoma	180	661	$3\,10^{-1}$	$-\infty$	3
lung cancer	197	1000	$2\,10^{-1}$	$-\infty$	4
breast cancer(A)	97	1213	$8\,10^{-2}$	$-\infty$	3
colon cancer	62	2000	$3\,10^{-2}$	$-\infty$	2
leukemia	72	3571	$2\,10^{-2}$	$-\infty$	2
brain cancer	42	5597	$7\,10^{-3}$	$-\infty$	5
breast cancer(W)	49	7129	$7\,10^{-3}$	$-\infty$	2
accent recognition	117	$5\,10^{5}$	$2\,10^{-4}$	$-\infty$	2

Table: *The last column is the number of classes in the pattern recognition task. Normally we need to compute the class condition covariance matrices.*

Empirical Framework for Predictive Analytics

```
For $r = 1$ to $R$
    Draw $l$ items without replacement from $\mathcal{D}$ to form $\mathcal{T}_r$
    Train $\hat{f}^{(r)}(\cdot)$ based on the $l$ items in $\mathcal{T}_r$
    Predict $\hat{f}^{(r)}(\mathbf{x}_i)$ for the $m$ items in $\mathcal{V}_r = \mathcal{D}\backslash\mathcal{T}_r$
    Calculate $\widehat{\texttt{EPMSE}}(\hat{f}^{(r)}) = \frac{1}{m}\sum_{\mathbf{z}_i \in \mathcal{V}_r} \ell(\mathrm{y}_i, \hat{f}^{(r)}(\mathbf{x}_i))$
End
```

- *Compute the average EPMSE for $\hat{f}$, namely*

$$\texttt{average}\{\texttt{EPMSE}(\hat{f})\} = \frac{1}{R}\sum_{r=1}^{R} \widehat{\texttt{EPMSE}}(\hat{f}^{(r)}).$$

Theoretical Aspects of Statistical Learning

With labels taken from $\{-1,+1\}$, *we have*

$$\hat{R}_{emp}(f) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}|\mathrm{y}_i - f(\mathbf{x}_i)| = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{\{\mathrm{y}_i \neq f(\mathbf{x}_i)\}}$$

For every $f \in \mathcal{F}$, *and* $n > h$, *with probability at least* $1-\eta$, *we have*

$$R(f) \leq \hat{R}_{emp}(f) + \sqrt{\frac{h\log\left(\frac{2n}{h}+1\right) - \log\left(\frac{4}{\eta}\right)}{n}}$$

In the above formula, h *is the VC (Vapnik-Chervonenkis) dimension of the space* $\mathcal{F}$ *of functions from which* f *is taken.*

Comparison of Learning Methods

	LDA	SVM	CART	rForest	GaussPR
Musk	*0.2227*	*0.1184*	*0.2450*	*0.1152*	*0.1511*
Pima	*0.2193*	*0.2362*	*0.2507*	*0.2304*	*0.2304*
Crabs	*0.0452*	*0.0677*	*0.1970*	*0.1097*	*0.0702*

	kNN	adaBoost	NeuralNet	Logistic
Musk	*0.1922*	*0.1375*	*0.1479*	*0.2408*
Pima	*0.3094*	*0.2243*	*0.2570*	*0.2186*
Crabs	*0.0938*	*0.1208*	*0.0350*	*0.0363*

Table: Computations made with *The Mighty R*

Comparison of Learning Methods

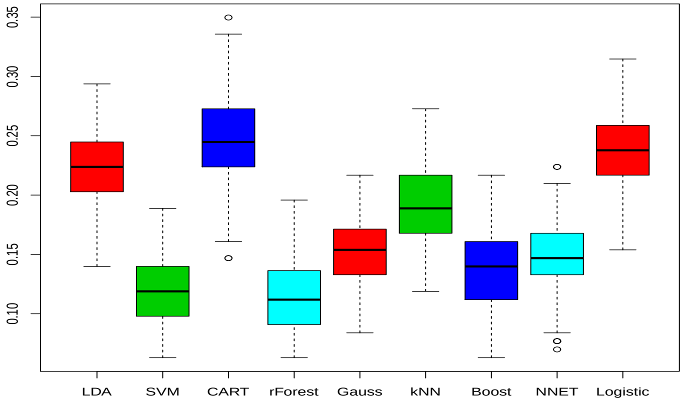


Figure: *Comparison of the average prediction error over* $R = 100$ *replications on the Musk data*

Comparison of Learning Methods

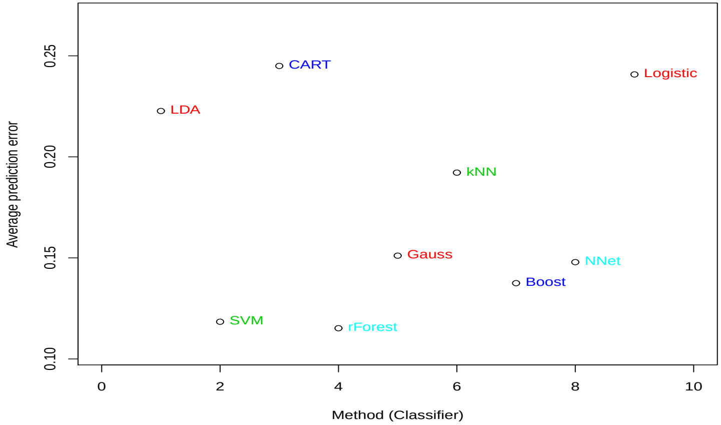


Figure: *Comparison of the average prediction error over $R = 100$ replications on the Musk data*

Comparison of Learning Methods

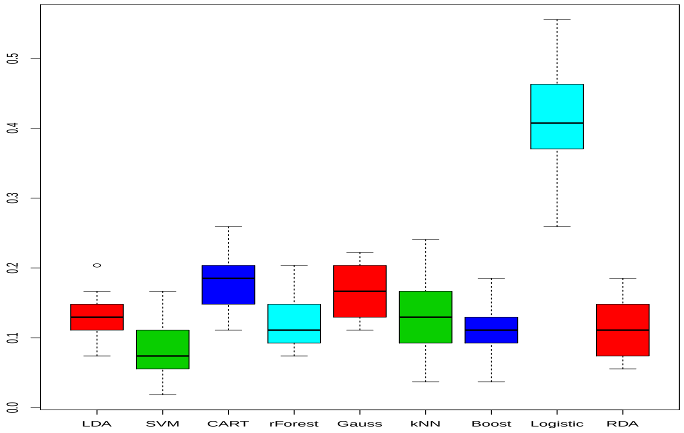


Figure: *Comparison of the average prediction error over $R = 100$ replications on the lymphoma data*

Comparison of Learning Methods

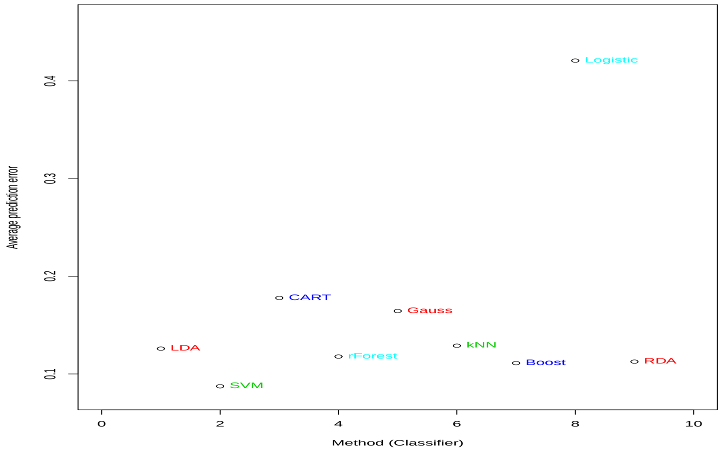


Figure: *Comparison of the average prediction error over $R = 100$ replications on the lymphoma data*

No Free Lunch Theorem

(No Free Lunch)

There is no learning method that is universally superior to all other methods on all datasets. In other words, if a learning method is presented with a data set whose inherent patterns violate its assumptions, then that learning method will under-perform.

The above no free lunch theorem basically says that there is no such thing as a universally superior learning method that outperforms all other methods on all possible data, no matter how sophisticated the method may appear to be.

Exercise session

1. *Consider the Crabs Leptograpsus dataset and perform a thorough SVM Classification analysis on it*
    - *Use the "C-svc" with a polynomial kernel and compare it to the performance obtained with the RBF kernel. Compare both training errors and test errors.*
    - *Discuss of the effect of the kernel on the performance of SVM classifiers*
    - *Generate the corresponding ROC curves and comment*
2. *Consider the German credit dataset and perform a thorough SVM Classification analysis on it*
3. *Consider the Wisconsin breast cancer dataset and perform a thorough SVM Classification analysis on it*
4. *Consider the email spam data set and perform kernel SVM classification on it. Compare your results to what was obtained with discriminant analysis and logistic regression. Do the same for the German credit data.*
5. *Provide your own dataset and perform a thorough SVM Classification analysis on it*