

## Optimization

- ▶ Many more complex models in statistics and machine learning do not have analytic solutions to ML estimators.
- ▶ In most models parameters are learned by some numerical optimization technique.

$$\min_{\theta} F(\theta)$$

- ▶ How many minima are there?
- ▶ How do we find optimal  $\theta$ ?
- ▶ Are we guaranteed to find the global optimum  $\theta^*$ , rather just a local one?
- ▶ How efficiently can we solve for  $\theta$ ?
- ▶ What if there are constraints?

196

## Constrained Optimization

- ▶ Optimization problems with constraints, e.g.

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g_i(\theta) \leq 0 \quad \text{for } i = 1, \dots, I \\ & h_j(\theta) = 0 \quad \text{for } j = 1, \dots, J \end{aligned}$$

where  $g_i$  enforce inequality constraints and  $h_j$  equality constraints.

- ▶ Can write this succinctly:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g(\theta) \preceq 0 \\ & h(\theta) = 0 \end{aligned}$$

where  $g : \mathbb{R}^d \rightarrow \mathbb{R}^I$  is a vector-valued function with  $g(\theta)_i = g_i(\theta)$ . Similarly  $h(\theta) : \mathbb{R}^d \rightarrow \mathbb{R}^J$ .  $x \preceq y$  iff  $x_i \leq y_i \forall i$ .

- ▶ These problems are called **programmes**.

197

## Constrained Optimization

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d} \quad & F(\theta) \\ \text{subject to} \quad & g(\theta) \preceq 0 \\ & h(\theta) = 0 \end{aligned}$$

- ▶ We can enforce constraints by using **Lagrange multipliers** or **dual variables**  $\lambda \in \mathbb{R}^I$  and  $\kappa \in \mathbb{R}^J$ .
- ▶ The optimization problem can be written as a mini-max optimization of the Lagrangian:

$$\min_{\theta} \max_{\lambda \geq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) = \min_{\theta} \max_{\lambda \geq 0, \kappa} F(\theta) + \lambda^\top g(\theta) + \kappa^\top h(\theta)$$

- ▶ Intuition: For any  $\theta$ , we have:

$$\max_{\lambda \geq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) = \begin{cases} +\infty & \text{if there is some unsatisfied constraint,} \\ F(\theta) & \text{if all constraints are satisfied.} \end{cases}$$

So the outer minimization over  $\theta$  results in the same optimization problem.

198

## Convex Optimization

- ▶ A function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  is **convex** if

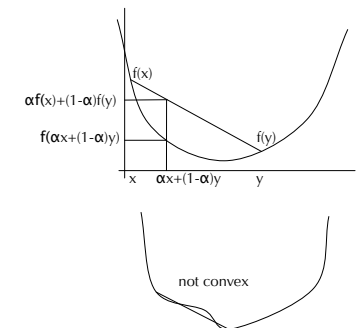
$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

for all  $x, y \in \mathbb{R}^d$ ,  $\alpha \in [0, 1]$ .

- ▶ For smooth functions: Equivalent to 2nd derivative (**Hessian**) being positive semidefinite.
- ▶ A programme is a **convex programme** if:
  - ▶  $F(\theta)$  is convex,
  - ▶  $g_i(\theta)$  is convex for each  $i$ ,
  - ▶  $h(\theta) = A\theta + b$  is affine.

- ▶ Examples: linear, quadratic, semidefinite programming.

- ▶ Convex programmes have a unique minimum (typically), which can be efficiently found.



199

## Convex Duality

- ▶ Say the minimum is  $p^*$ , and occurred at  $\theta^*$ .
- ▶ The **dual programme** inverts the order of max and min:

$$p^* = \min_{\theta} \max_{\lambda \geq 0, \kappa} \mathcal{L}(\theta, \lambda, \kappa) \geq \max_{\lambda \geq 0, \kappa} \min_{\theta} \mathcal{L}(\theta, \lambda, \kappa) = d^*$$

where the dual optimum is  $d^*$ .

- ▶ **Karush-Kuhn-Tucker Theorem:** Subject to regularity conditions, a solution  $\theta^*$  is the optimal solution of a convex programme, if and only if there are  $\lambda^*$  and  $\kappa^*$  (the dual optimal solution) such that:

- ▶ **Primal feasible:**  $g(\theta^*) \leq 0, h(\theta^*) = 0$ .
- ▶ **Dual feasible:**  $\lambda^* \geq 0$ .
- ▶  $(\theta^*, \lambda^*, \kappa^*)$  is a **saddle point** of  $\mathcal{L}$ : For every  $\theta, \lambda \geq 0, \kappa$ , we have

$$\mathcal{L}(\theta^*, \lambda, \kappa) \leq \mathcal{L}(\theta^*, \lambda^*, \kappa^*) \leq \mathcal{L}(\theta, \lambda^*, \kappa^*)$$

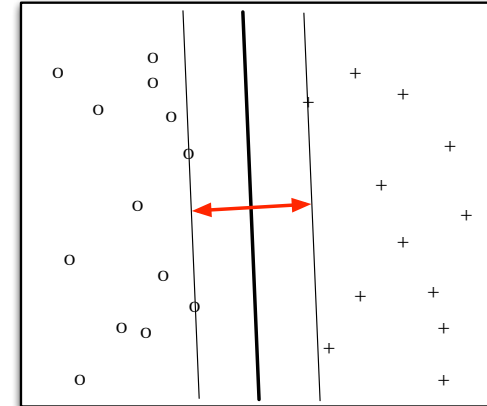
- ▶  $\nabla_{\theta} \mathcal{L}(\theta^*, \lambda^*, \kappa^*) = \nabla_{\theta} F(\theta^*) + (\lambda^*)^{\top} \nabla_{\theta} g(\theta^*) + (\kappa^*)^{\top} \nabla_{\theta} h(\theta^*) = 0$
- ▶ **Complementary slackness:** For every  $i$ ,

$$\lambda_i^* g_i(\theta^*) = 0$$

200

## Maximum Margin Classification

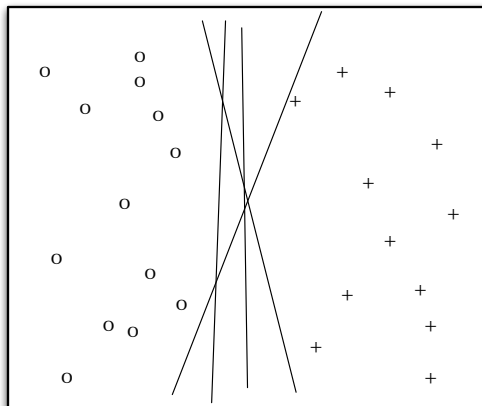
- ▶ Good choice of separating hyperplane: one with **large margin**.
- ▶ Such a hyperplane will be defined by a number of data vectors close to the boundary—the **support vectors**, leading to a method called **support vector machines**.



202

## Linear Classification

- ▶ A dataset with  $\{+1, -1\}$  labels is **linearly separable** if there is a hyperplane separating two classes.
- ▶ Typically there will be an infinite number of such **separating hyperplanes**.



201

## Support Vector Machines

- ▶ A hyperplane can be parametrized as:

$$g(x) = a + b^{\top} x = 0$$

with the classification given by  $\text{sign}(g(x))$ .

- ▶ Distance and classification of a point  $x_i$  from hyperplane is  $g(x_i)/\|b\|$ .
- ▶ Multiplying  $a$  and  $b$  by  $c > 0$  does not affect result. Rescale such that margin (closest distance of data vectors to hyperplane) is  $1/\|b\|$ .

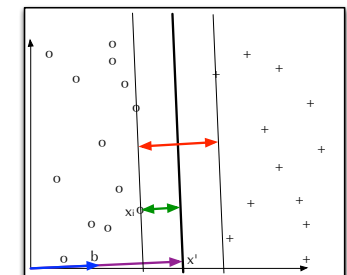
$$y_i(a + b^{\top} x_i)/\|b\| \geq 1/\|b\|$$

$$y_i(a + b^{\top} x_i) \geq 1$$

- ▶ Constrained optimization problem to solve for  $a, b$ :

$$\max_{a,b} 1/\|b\| \quad \Leftrightarrow \quad \min_{a,b} \frac{1}{2} \|b\|^2$$

subject to  $y_i(a + b^{\top} x_i) \geq 1$       subject to  $y_i(a + b^{\top} x_i) \geq 1$  for all  $i$



203

## Support Vector Machines

- ▶ Introduce Lagrange multipliers  $\lambda_i \geq 0$  to enforce constraints:

$$\min_{a,b} \max_{\lambda \geq 0} \mathcal{L}(a, b, \lambda) = \frac{1}{2} \|b\|^2 + \sum_{i=1}^n \lambda_i (1 - y_i(a + b^\top x_i))$$

- ▶ KKT optimality conditions:

Zero derivatives:  $\nabla_a \mathcal{L}(a^*, b^*, \lambda^*) = - \sum_{i=1}^n \lambda_i^* y_i = 0$

$$\nabla_b \mathcal{L}(a^*, b^*, \lambda^*) = b^* - \sum_{i=1}^n \lambda_i^* y_i x_i = 0$$

Primal feasibility:  $y_i(a^* + (b^*)^\top x_i) \geq 1$

Dual feasibility:  $\lambda_i^* \geq 0$

Complementary slackness:  $\lambda_i^* (1 - y_i(a^* + (b^*)^\top x_i)) = 0$

204

## Support Vector Machines

- ▶ Substituting optimal  $a^*$  and  $b^*$  into Lagrangian leads to the **dual optimization problem**:

$$\max_{\lambda} \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j (x_i)^\top (x_j)$$

$$\text{subject to } \sum_{i=1}^n \lambda_i y_i = 0$$

$$\lambda \geq 0$$

A **quadratic programme**. Standard solvers can be used to find optimal  $\lambda^*$  in  $O(n^3)$  cost.

- ▶ Those vectors with  $\lambda_i > 0$  are called **support vectors**.
- ▶ Complementary slackness implies that if  $x_i$  does not lie on boundary, then  $\lambda_i = 0$ , i.e. not a support vector.
- ▶ Discriminant function is

$$g(x) = a^* + \sum_{i=1}^n \lambda_i^* y_i x_i^\top x$$

where  $a^*$  can be solved by noting that  $y_j g(x_j) = 1$  for a support vector  $x_j$ .

205

## Soft-Margin Support Vector Machines

- ▶ For **non-linearly separable** datasets, we can allow for **margin violations**

$$\xi_i = \begin{cases} 1 - y_i(a + b^\top x_i) & \text{if margin violated,} \\ 0 & \text{if not violated.} \end{cases}$$

$$= \max(0, 1 - y_i(a + b^\top x_i))$$

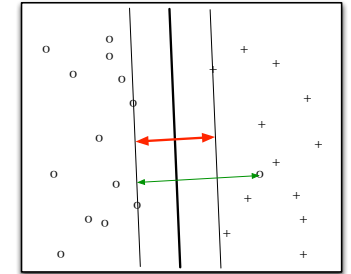
- ▶ Penalizing violations by their magnitude,

$$\min_{a,b,\xi} \frac{1}{2} \|b\|^2 + C \sum_{i=1}^n \xi_i$$

subject to  $y_i(a + b^\top x_i) \geq 1 - \xi_i$

$$\xi_i \geq 0$$

where  $C$  is a tuning parameter.



206

## Soft-Margin Support Vector Machines

- ▶ Introduce Lagrange multipliers  $\lambda_i \geq 0, \gamma_i \geq 0$  to enforce constraints:

$$\mathcal{L}(a, b, \xi, \lambda, \gamma) = \frac{1}{2} \|b\|^2 + C \sum_{i=1}^n \xi_i + \sum_{i=1}^n \lambda_i (1 - \xi_i - y_i(a + b^\top x_i)) - \sum_{i=1}^n \gamma_i \xi_i$$

- ▶ KKT optimality conditions:

Zero derivatives:  $\nabla_a \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = - \sum_{i=1}^n \lambda_i^* y_i = 0$

$$\nabla_b \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = b^* - \sum_{i=1}^n \lambda_i^* y_i x_i = 0$$

$$\nabla_{\xi} \mathcal{L}(a^*, b^*, \xi^*, \lambda^*, \gamma^*) = C - \lambda_i^* - \gamma_i^* = 0$$

Primal feasibility:  $y_i(a^* + (b^*)^\top x_i) \geq 1 - \xi_i^*$

$$\xi_i^* \geq 0$$

Dual feasibility:  $\lambda_i^* \geq 0$

$$\gamma_i^* \geq 0$$

Complementary slackness:  $\lambda_i^* (1 - \xi_i^* - y_i(a^* + (b^*)^\top x_i)) = 0$

$$\gamma_i^* \xi_i^* = 0$$

207

## Soft-Margin Support Vector Machines

- ▶ Setting derivatives of primal variables to zero leads to the dual programme:

$$\begin{aligned} \max_{\lambda} \quad & \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j (x_i)^\top (x_j) \\ \text{subject to} \quad & \sum_{i=1}^n \lambda_i y_i = 0 \\ & 0 \leq \lambda_i \leq C \end{aligned}$$

Only difference is the **box constraint** on  $\lambda_i \in [0, C]$ .

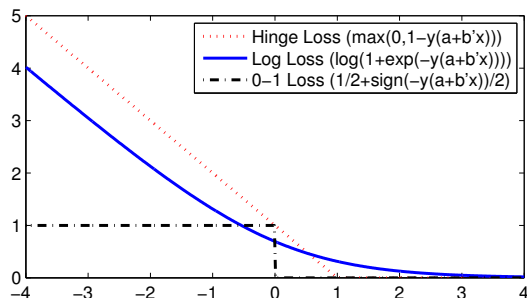
208

## Soft-Margin Support Vector Machines

- ▶ From primal programme, we can first minimize over  $\xi_i$ 's, leading to an unconstrained convex programme:

$$\min_{a,b} \quad \frac{1}{2} \|b\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i(a + b^\top x_i))$$

- ▶ Interpretation: regularized empirical risk minimization with the **hinge loss**.



209

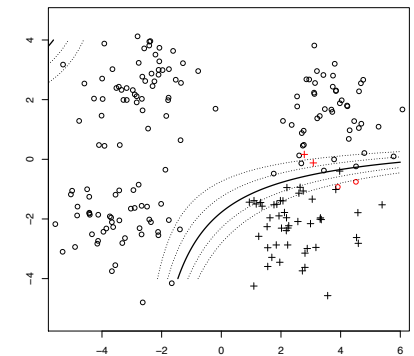
## Support Vector Machines – Discussion

- ▶ **Multiclass classification:** If there are more than two classes, there are multiclass generalizations of the SVM.
- ▶ A simple practical idea: treat a multiclass problem as multiple binary classification problems.
  - ▶ One-vs-one: train  $K(K-1)$  binary SVMs, for each pair of classes. At test time, predict class that got the most votes.
  - ▶ One-vs-rest: train  $K$  binary SVMs, one for each class vs all other classes. At test time, predict class with largest discriminant value  $a_k + b_k^\top x$ .
- ▶ Optimization for large scale problems:
  - ▶ Standard quadratic programme solvers not scalable.
  - ▶ Sequential minimal optimization (SMO): iterative solve pairs of  $\lambda_i$ 's.
  - ▶ Pegasos : stochastic gradient descent on regularized hinge loss objective.
- ▶  $L_2$  regularization controls overfitting.
- ▶ Not probabilistic and cannot produce uncertainty estimates.
- ▶ Statistical learning theory foundations.
- ▶ Further readings:
  - ▶ Bishop, Chapter 6.
  - ▶ Christopher Burgess, A Tutorial on Support Vector Machines for Pattern Recognition. 1998.

210

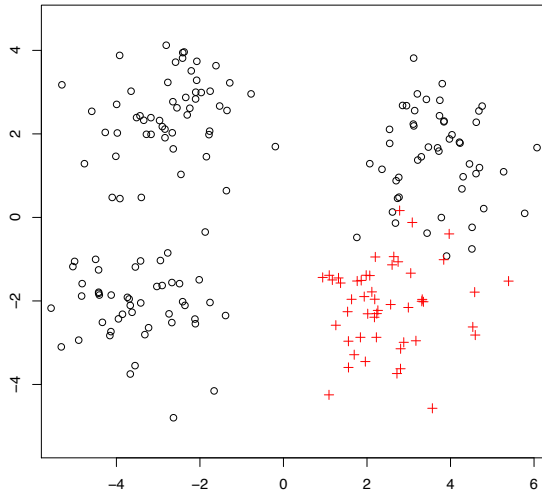
## Nonlinear Methods

- ▶ Decision boundaries and regression functions often need to be nonlinear.
- ▶ One general approach: transform data  $x \mapsto \phi(x)$ .
- ▶ A **global** approach. Decisions and optimal parameters depend on **whole** training dataset.
- ▶ Alternative approach:  $p(Y = 1|X = x)$  or  $f(x)$  depends only on data cases in **local neighbourhood** of  $x$ .



211

## Local Methods



212

## k-Nearest Neighbours

- ▶ Nearest neighbours are simple and essentially model-free methods for classification.
- ▶ Weaker modelling assumptions than e.g. LDA, Naïve Bayes and logistic regression.
- ▶ These methods are not very useful for understanding relationships between attributes and class predictions.
- ▶ As **black box** classification methods however, they are often perform reasonably on real life problems and provide a good benchmark.
- ▶ Can break down in high-dimensional data:
  - ▶ Effectively, partitions input space into regions each containing  $k$  data points, and prediction in each region estimated separately.
  - ▶ In a space of dimension  $p \gg 0$ , number of regions needed is  $R = m^p$ , so size of dataset needed is  $km^p$ .

214

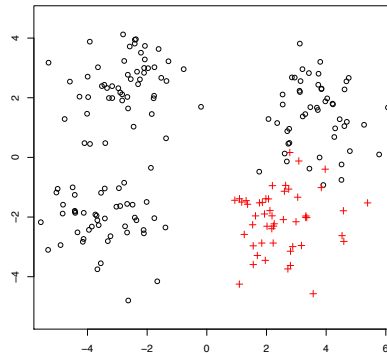
## k-Nearest Neighbours

- ▶ A simple, local, nonlinear, non-model-based, method.
- ▶ Prediction at a data vector  $x$  is simply determined by the  $k$  nearest neighbours  $ne_k(x)$  of  $x$  among the training set.
- ▶ Classification: predict the **majority vote** of the neighbours:

$$f^{\text{kNN}}(x) = \underset{l}{\operatorname{argmax}} \quad |\{j \in ne_k(x) : y_j = l\}|.$$

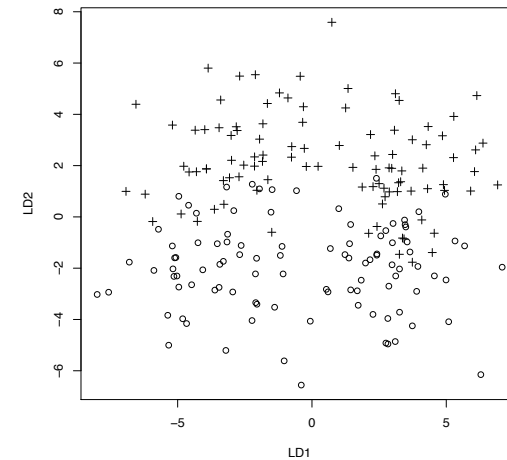
- ▶ Regression: predict the average among the neighbours:

$$f^{\text{kNN}}(x) = \frac{\sum_{j \in ne_k(x)} y_j}{\sum_{j \in ne_k(x)} 1}.$$



213

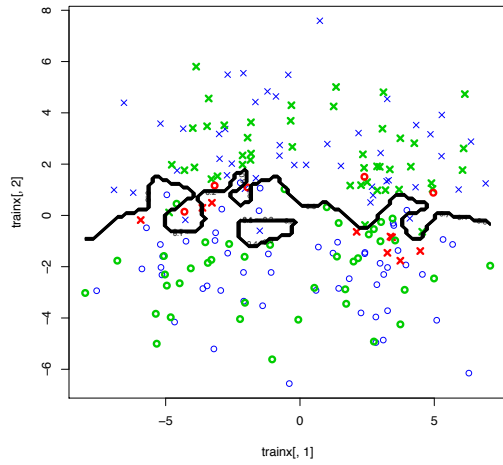
## k-Nearest Neighbour Demo



Data

215

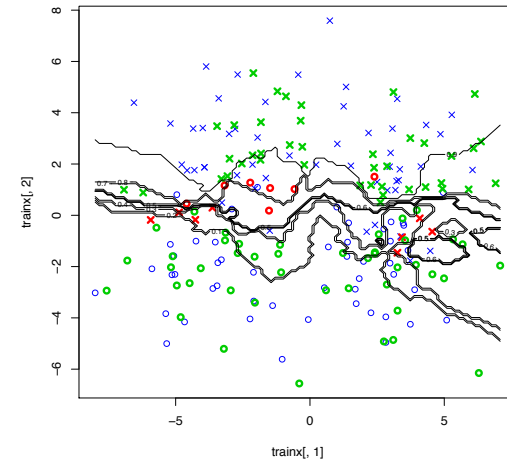
## k-Nearest Neighbour Demo



Result of 1NN

216

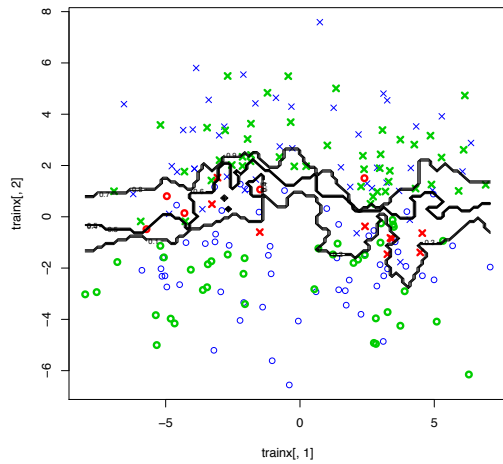
## k-Nearest Neighbour Demo



Result of 5NN

218

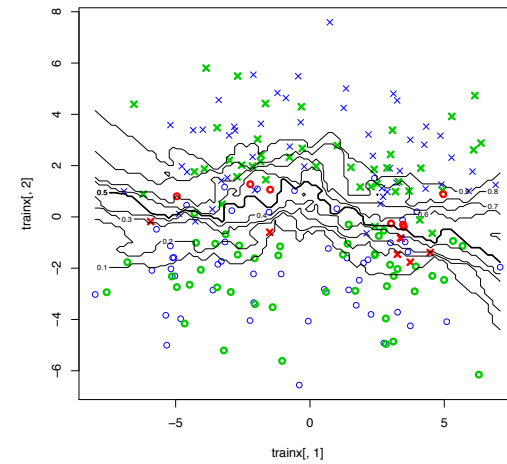
## k-Nearest Neighbour Demo



Result of 3NN

217

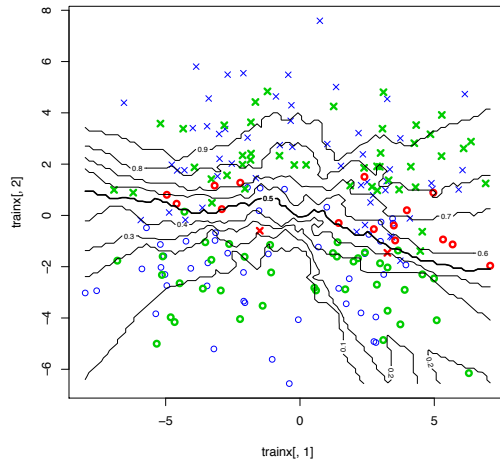
## k-Nearest Neighbour Demo



Result of 11NN

219

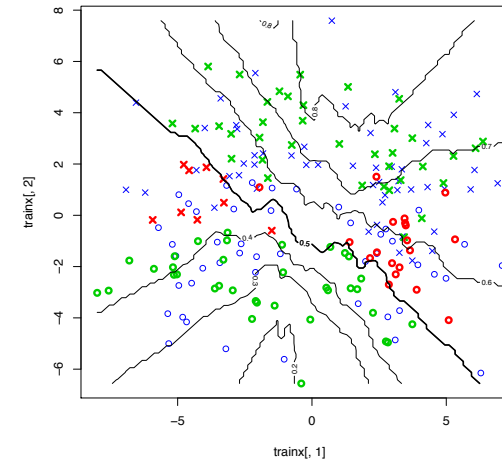
## k-Nearest Neighbour Demo



Result of 21NN

220

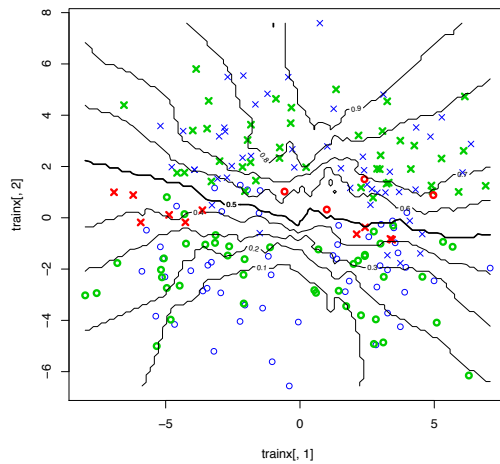
## k-Nearest Neighbour Demo



Result of 51NN

222

## k-Nearest Neighbour Demo



Result of 31NN

221

## k-Nearest Neighbour Demo – R Code I

```
library(MASS)
## load crabs data
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
## project into first two LD
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- as.matrix(cb.ldp[,1:2])
y <- as.numeric(crabs[,2])-1
x <- x + rnorm(dim(x)[1]*dim(x)[2])*1.5
eqsplot(x,pch=2+y+1,col=1)

k <- 3

kNN <- function(k,x,y,gridsize=100) {
  n <- length(y)
  p <- dim(x)[2]
  i <- sample(rep(c(TRUE,FALSE),each=n/2),n,replace=FALSE)
  train <- (1:n)[i]
  test <- (1:n)[!i]
  trainx <- x[train,]
  trainy <- y[train]
  testx <- x[test,]
  testy <- y[test]

  trainn <- dim(trainx)[1]
  testn <- dim(testx)[1]

  gridx1 <- seq(min(x[,1]),max(x[,1]),length=gridsize)
  gridx2 <- seq(min(x[,2]),max(x[,2]),length=gridsize)
  gridx <- as.matrix(expand.grid(gridx1,gridx2))
  gridn <- dim(gridx)[1]

  # calculate distances, smart and intelligently.
  trainxx <- t((trainx*trainx) %*% matrix(1,p,1))
```

223

## k-Nearest Neighbour Demo – R Code II

```

testxx <- (testx+testx)  %*% matrix(1,p,1)
gridxx  <- (gridx+gridx) %*% matrix(1,p,1)
testtraindist <- matrix(1,testn,1) %*% trainxx +
  testxx %*% matrix(1,1,trainn) -
  2*(testx %*% t(trainx))
gridtraindist <- matrix(1,gridn,1) %*% trainxx +
  gridxx %*% matrix(1,1,trainn) -
  2*(gridx %*% t(trainx))

# predict
testp <- numeric(testn)
gridp <- numeric(gridn)
for (j in 1:testn) {
  nearestneighbors <- order(testtraindist[j,])[1:k]
  testp[j] <- mean(trainy[nearestneighbors])
}
for (j in 1:gridn) {
  nearestneighbors <- order(gridtraindist[j,])[1:k]
  gridp[j] <- mean(trainy[nearestneighbors])
}
predy <- as.numeric(testp>.5)

plot(trainx[,1],trainx[,2],pch=trainy*3+1,col=4,lwd=.5)
points(testx[,1],testx[,2],pch=testy*3+1,col=2+(predy==testy),lwd=3)
contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
  levels=seq(.1,.9,.1),lwd=.5,add=TRUE)
contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
  levels=c(.5),lwd=2,add=TRUE)
}

```

224

## K-Nearest Neighbours – Discussion

- ▶ kNN is sensitive to distances: normalize data and find suitable metric.
- ▶ Choice of  $k$  important: controls flexibility of model.
- ▶ Computational cost of kNN is very high.
  - ▶ Need to store **all** training data.
  - ▶ Need to compare each test data vector to **all** training data.
  - ▶ Need **a lot of data** in high dimensions.
- ▶ Mitigation techniques:
  - ▶ Compute approximate nearest neighbours, using kd-trees, cover trees, random forests.
  - ▶ Apply K-means to data in each class, to reduce size of data (need to use large  $K$ ).

226

## Asymptotic Performance of 1NN

- ▶ Let  $(x_i, y_i)_{i=1}^n$  be training data where  $x_i \in \mathbb{R}^p$  and  $y_i \in \{1, 2, \dots, K\}$ .
- ▶ We define

$$\hat{y}_{\text{Bayes}}(x) = \arg \max_{l \in \{1, \dots, K\}} \pi_l f_l(x)$$

and

$$\hat{y}_{1\text{NN}}(x) = y \text{ (nearest neighbour of } x \text{)}.$$

- ▶ The (optimal) Bayes risk and 1NN risk are:

$$R_{\text{Bayes}} = \mathbb{E} [\mathbb{I}(Y \neq \hat{y}_{\text{Bayes}}(X))]$$

$$R_{1\text{NN}} = \mathbb{E} [\mathbb{I}(Y \neq \hat{y}_{1\text{NN}}(X))]$$

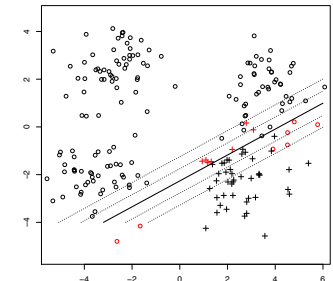
- ▶ As  $n \rightarrow \infty$ , we have the following powerful result

$$R_{\text{Bayes}} \leq R_{1\text{NN}} \leq 2R_{\text{Bayes}} - \frac{K}{K-1} R_{\text{Bayes}}^2.$$

225

## Non-linear Problems

- ▶ Linear methods (PCA, LDA, linear and logistic regression) are simple and effective techniques to learn from data “to first order”.
- ▶ To capture more intricate information from data, flexible, non-linear methods are often needed.
  - ▶ Explicit non-linear transformations  $x \mapsto \phi(x)$ .
  - ▶ Local methods like kNN.
- ▶ **Kernel methods**: introduce non-linearities through **implicit** non-linear transforms, often local in nature.



227



## The Kernel Method

- ▶ Back to the soft-margin SVM. The dual objective is:

$$\max_{\lambda} \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j \phi(x_i)^\top \phi(x_j) \quad \text{subject to} \quad \begin{cases} \sum_{i=1}^n \lambda_i y_i = 0 \\ 0 \leq \lambda \leq C \end{cases}$$

- ▶ Suppose  $p = 2$ , and we would like to introduce quadratic non-linearities,

$$\phi(x_i) = (1, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, x_{i1}^2, x_{i2}^2, x_{i1}x_{i2})^\top$$

Then

$$\begin{aligned} \phi(x_i)^\top \phi(x_j) &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + x_{i1}x_{i2}x_{j1}x_{j2} \\ &= (1 + x_i^\top x_j)^2 \end{aligned}$$

- ▶ Since only dot-products are needed in the objective function, non-linear transform need not be computed explicitly!
- ▶ Generally,  $m$ -order interactions can be implemented simply by  $\phi(x_i)^\top \phi(x_j) = (1 + x_i^\top x_j)^m$ . This is called a **polynomial kernel**.

228

## The Kernel Method

- ▶ The **Gram matrix** is the matrix of dot-products,  $B_{ij} = \phi(x_i)^\top \phi(x_j)$ .

$$B = \begin{pmatrix} -\phi(x_1)^\top - \\ \vdots \\ -\phi(x_i)^\top - \\ \vdots \\ -\phi(x_n)^\top - \end{pmatrix} \times \begin{pmatrix} | & & | & & | \\ \phi(x_1) & \cdots & \phi(x_j) & \cdots & \phi(x_n) \\ | & & | & & | \end{pmatrix}$$

- ▶ Since  $B = \Phi\Phi^\top$ , it is symmetric and positive semidefinite.
- ▶ The Gram matrix is sufficient for training the soft-margin SVM.

$$\max_{\lambda} \sum_{i=1}^n \lambda_i - \frac{1}{2} \sum_{i,j=1}^n \lambda_i \lambda_j y_i y_j B_{ij} \quad \text{subject to} \quad \begin{cases} \sum_{i=1}^n \lambda_i y_i = 0 \\ 0 \leq \lambda \leq C \end{cases}$$

229

## The Kernel Method

- ▶ A **kernel** is a function  $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  such that:

- ▶ Symmetric:  $\kappa(x, x') = \kappa(x', x)$ .
- ▶ Positive semidefinite: given any finite set  $\{x_i\}_{i=1}^n \subset \mathcal{X}$ , the matrix  $B \in \mathbb{R}^{n \times n}$  with entries  $B_{ij} = \kappa(x_i, x_j)$  is positive definite. Equivalently, for any  $c \in \mathbb{R}^n$ ,

$$\sum_{i=1}^n \sum_{j=1}^n c_i c_j \kappa(x_i, x_j) \geq 0$$

- ▶ **Mercer's Theorem:** if  $\kappa$  is continuous, symmetric and positive semidefinite, then there is a function  $\phi : \mathcal{X} \rightarrow \mathcal{H}$  into a Hilbert space  $\mathcal{H}$  with inner product  $\langle \cdot, \cdot \rangle$  such that

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$$

230

## The Kernel Method

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$$

- ▶ We do not need to compute the features ever—the Gram matrix is sufficient for learning and prediction. The discriminant function (absorbing  $a$  into  $b$ ) is

$$g(x) = \sum_{i=1}^n \lambda_i^* y_i \phi(x_i)^\top \phi(x) = \sum_{i=1}^n \lambda_i^* y_i \kappa(x_i, x)$$

- ▶ The function  $\phi$  can be interpreted as non-linear **features** of our data vectors  $x \in \mathcal{X}$ .
- ▶ Generally, the Hilbert space can be **infinite-dimensional**, so we are effectively computing an infinite number of features of our data, and learning a SVM based on all features.
- ▶ There are an infinite number of parameters in the SVM—a **nonparametric** method.
- ▶ The  $L_2$  regularization of SVMs is very important to prevent overfitting.

231

## Examples of Kernels

- ▶ Polynomial kernel:

$$\kappa(x, x') = (1 + x^\top x')^m$$

- ▶ **Gaussian, radial-basis function (RBF), or squared-exponential kernel:**

$$\kappa(x, x') = \exp\left(-\frac{1}{2}\|x - x'\|_M^2\right)$$

This leads to a discriminant function of form

$$g(x) = \sum_{i=1}^n \lambda_i^* y_i \exp\left(-\frac{1}{2}\|x_i - x\|_M^2\right)$$

A local method very similar to kNN.

- ▶ If  $\kappa_1$  and  $\kappa_2$  are both kernels, then so are kernels defined by

$$\kappa_3(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$$

$$\kappa_4(x, x') = \kappa_1(x, x') \times \kappa_2(x, x')$$

## Kernel Methods – Discussion

- ▶ The kernel method allows for very flexible and powerful machine learning models.
- ▶ Kernels can be defined over much more complex structures than vectors, e.g. graphs, strings.
- ▶ Can be hard to interpret.
- ▶  $O(n^3)$  computation and  $O(n^2)$  memory cost can be prohibitive.
- ▶ Further readings:
  - ▶ Bishop, Chapter 6.
  - ▶ Christopher Burgess, A Tutorial on Support Vector Machines for Pattern Recognition. 1998.
  - ▶ Rasmussen and Williams, Gaussian Processes for Machine Learning. 2006.

232

234

## Kernel SVM Demo

```
library(MASS)
library(e1071)
## load crabs data, project onto LD space, add noise.
data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldap <- predict(cb.lda)
x <- as.matrix(cb.ldap[,1:2])
y <- as.numeric(crabs[,2])-1
x <- x + rnorm(dim(x)[1]*dim(x)[2])*1.5
gridsize <- 100
xlim <- c(min(x[,1]),max(x[,1]))
ylim <- c(min(x[,2]),max(x[,2]))
gridx1 <- seq(xlim[1],xlim[2],length=gridsize)
gridx2 <- seq(ylim[1],ylim[2],length=gridsize)
gridx <- as.matrix(expand.grid(gridx1,gridx2))
gridn <- dim(gridx)[1]
plot(x,pch=2+y+1,col=1,xlim=xlim,ylim=ylim)

n <- length(y)
p <- dim(x)[2]
i <- sample(rep(c(TRUE,FALSE),each=n/2),n,replace=FALSE)
train <- (1:n)[i]
test <- (1:n)[!i]
trainx <- x[train,]
trainy <- y[train]
testx <- x[test,]
testy <- y[test]

svmdemo <- function(kernel,gamma=1,coef0=0,cost=1,degree=3) {
  model <- svm(trainx,trainy,kernel=kernel,gamma=gamma,coef0=coef0,degree=degree,cost=cost)
  gridp <- predict(model,gridx)
  predy <- as.numeric(predict(model,testx)>.5)

  plot(trainx[,1],trainx[,2],pch=trainy+3+1,col=4,lwd=.5,xlim=xlim,ylim=ylim)
  points(testx[,1],testx[,2],pch=testy+3+1,col=2+(predy==testy),lwd=3)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),levels=seq(.1,.9,.1),lwd=.5,add=TRUE)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),levels=c(.5),lwd=2,add=TRUE)
}
```

233