Kernel Methods for Learning

John Shawe-Taylor
Department of Computer Science
University College London
jst@cs.ucl.ac.uk

UDRC Summer School, University of Surrey

July 2015

Aim:

The course is intended to give an overview of the kernel approach to pattern analysis. This will cover:

- Why linear pattern functions?
- Why kernel approach?
- How to plug and play with the different components of a kernel-based pattern analysis system?

What won't be included:

- Other approaches to Pattern Analysis
- Complete History
- Bayesian view of kernel methods
- Most recent developments

OVERALL STRUCTURE

Part 1: Introduction to the Kernel methods approach.

Part 2: Projections and subspaces in the feature space.

Part 3: Stability of Pattern Functions with the example of Support Vector Machines.

Part 4: Other learning algorithms: novelty detection, boosting and multiple kernel learning.

Part 5: Kernel design strategies.

Part 1

- Kernel methods approach
- Worked example of kernel Ridge Regression
- Properties of kernels.

Kernel methods

Kernel methods (re)introduced in 1990s with Support Vector Machines

- Linear functions but in high dimensional spaces equivalent to non-linear functions in the input space
- Statistical analysis showing large margin can overcome curse of dimensionality
- Extensions rapidly introduced for many other tasks other than classification

Kernel methods approach

- Data embedded into a Euclidean feature (or Hilbert) space
- Linear relations are sought among the images of the data
- Algorithms implemented so that only require inner products between vectors
- Embedding designed so that inner products of images of two points can be computed directly by an efficient ‘short-cut’ known as the kernel.

Worked example: Ridge Regression

Consider the problem of finding a homogeneous real-valued linear function

$$g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle = \mathbf{x}'\mathbf{w} = \sum_{i=1}^{n} w_i x_i,$$

that best interpolates a given training set

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$$

of points $\mathbf{x}_i$ from $X \subseteq \mathbb{R}^n$ with corresponding labels y_i in $Y \subseteq \mathbb{R}$.

Possible pattern function

- Measures discrepancy between function output and correct output – squared to ensure always positive:

  $$f_g((\mathbf{x}, y)) = (g(\mathbf{x}) - y)^2$$

  Note that the pattern function f_g is not itself a linear function, but a simple functional of the linear functions g.

- We introduce notation: matrix $\mathbf{X}$ has rows the m examples of S. Hence we can write

  $$\xi = \mathbf{y} - \mathbf{X}\mathbf{w}$$

  for the vector of differences between $g(\mathbf{x}_i)$ and y_i.

Optimising the choice of g

Need to ensure flexibility of g is controlled – controlling the norm of $\mathbf{w}$ proves effective:

$$\min_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w}, S) = \min_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 + \|\xi\|^2,$$

where we can compute

$$\begin{aligned} \|\xi\|^2 &= \langle \mathbf{y} - \mathbf{X}\mathbf{w}, \mathbf{y} - \mathbf{X}\mathbf{w} \rangle \\ &= \mathbf{y}'\mathbf{y} - 2\mathbf{w}'\mathbf{X}'\mathbf{y} + \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} \end{aligned}$$

Setting derivative of $\mathcal{L}_\lambda(\mathbf{w}, S)$ equal to 0 gives

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n)\,\mathbf{w} = \mathbf{X}'\mathbf{y}$$

Primal solution

We get the primal solution weight vector:

$$\mathbf{w} = \left(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n\right)^{-1}\mathbf{X}'\mathbf{y}$$

and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}'\left(\mathbf{X}'\mathbf{X} + \lambda\mathbf{I}_n\right)^{-1}\mathbf{X}'\mathbf{y}$$

Dual solution

A dual solution should involve only computation of inner products – this is achieved by expressing the weight vector as a linear combination of the training examples:

$$\mathbf{X}'\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = \mathbf{X}'\mathbf{y} \quad \text{implies}$$

$$\mathbf{w} = \frac{1}{\lambda}(\mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{X}\mathbf{w}) = \mathbf{X}'\frac{1}{\lambda}(\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{X}'\alpha,$$

where

$$\alpha = \frac{1}{\lambda}(\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{1}$$

or equivalently

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i$$

Dual solution

Substituting $\mathbf{w} = \mathbf{X}'\alpha$ into equation (1) we obtain:

$$\lambda\alpha = \mathbf{y} - \mathbf{X}\mathbf{X}'\alpha$$

implying

$$\left(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m\right)\alpha = \mathbf{y}$$

This gives the dual solution:

$$\alpha = \left(\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}_m\right)^{-1}\mathbf{y}$$

and regression function

$$g(\mathbf{x}) = \mathbf{x}'\mathbf{w} = \mathbf{x}'\mathbf{X}'\alpha = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Key ingredients of dual solution

Step 1: Compute

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}'$ that is $\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$

Step 2: Evaluate on new point $\mathbf{x}$ by

$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Important observation: Both steps only involve inner products

Applying the ‘kernel trick’

Since the computation only involves inner products, we can substitute for all occurrences of $\langle \cdot, \cdot \rangle$ a kernel function κ that computes:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

and we obtain an algorithm for ridge regression in the feature space F defined by the mapping

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in F$$

Note if ϕ is the identity this remains in the input space.

A simple kernel example

The simplest non-trivial kernel function is the quadratic kernel:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^2$$

involving just one extra operation. But surprisingly this kernel function now corresponds to a complex feature mapping:

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}'\mathbf{z})^2 = \mathbf{z}'(\mathbf{x}\mathbf{x}')\mathbf{z} \\ &= \langle \mathrm{vec}(\mathbf{z}\mathbf{z}'), \mathrm{vec}(\mathbf{x}\mathbf{x}') \rangle\end{aligned}$$

where $\mathrm{vec}(A)$ stacks the columns of the matrix A on top of each other. Hence, κ corresponds to the feature mapping

$$\phi : \mathbf{x} \longmapsto \mathrm{vec}(\mathbf{x}\mathbf{x}')$$

Implications of the kernel trick

- Consider for example computing a regression function over 1000 images represented by pixel vectors – say $32 \times 32 = 1024$.
- By using the quadratic kernel we implement the regression function in a $1,000,000$ dimensional space
- but actually using less computation for the learning phase than we did in the original space.

Implications of kernel algorithms

- Can perform linear regression in very high-dimensional (even infinite dimensional) spaces efficiently.

- This is equivalent to performing non-linear regression in the original input space: for example quadratic kernel leads to solution of the form
$$g(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle^2$$
that is a quadratic polynomial function of the components of the input vector $\mathbf{x}$.

- Using these high-dimensional spaces must surely come with a health warning, what about the curse of dimensionality?

Part 2

- Simple classification algorithm
- Principal components analysis.
- Kernel canonical correlation analysis.

Simple classification algorithm

- Consider finding the centres of mass of positive and negative examples and classifying a test point by measuring which is closest

$$h(\mathbf{x}) = \operatorname{sgn}\left(\|\phi(\mathbf{x}) - \phi_{S_-}\|^2 - \|\phi(\mathbf{x}) - \phi_{S_+}\|^2\right)$$

- we can express as a function of kernel evaluations

$$h(\mathbf{x}) = \operatorname{sgn}\left(\frac{1}{m_+}\sum_{i=1}^{m_+}\kappa(\mathbf{x},\mathbf{x}_i) - \frac{1}{m_-}\sum_{i=m_++1}^{m}\kappa(\mathbf{x},\mathbf{x}_i) - b\right),$$

where

$$b = \frac{1}{2m_+^2}\sum_{i,j=1}^{m_+}\kappa(\mathbf{x}_i,\mathbf{x}_j) - \frac{1}{2m_-^2}\sum_{i,j=m_++1}^{m}\kappa(\mathbf{x}_i,\mathbf{x}_j)$$

Simple classification algorithm

- equivalent to dividing the space with a hyperplane perpendicular to the line half way between the two centres with vector given by

$$\mathbf{w} = \frac{1}{m^+}\sum_{i=1}^{m^+}\phi(\mathbf{x}_i) - \frac{1}{m^-}\sum_{i=m^++1}^{m}\phi(\mathbf{x}_i)$$

- Function is the difference in likelihood of the Parzen window density estimators for positive and negative examples

- We will see some examples of the performance of this algorithm in a moment.

Variance of projections

- Consider projections of the datapoints $\phi(\mathbf{x}_i)$ onto a unit vector direction $\mathbf{v}$ in the feature space: average is given by

$$\mu_{\mathbf{v}} = \hat{\mathrm{E}}\left[\|P_{\mathbf{v}}(\phi(\mathbf{x}))\|\right] = \hat{\mathrm{E}}\left[\mathbf{v}'\phi(\mathbf{x})\right] = \mathbf{v}'\phi_S$$

  of course this is 0 if the data has been centred.

- average squared is given by

$$\hat{\mathrm{E}}\left[\|P_{\mathbf{v}}(\phi(\mathbf{x}))\|^2\right] = \hat{\mathrm{E}}\left[\mathbf{v}'\phi(\mathbf{x})\phi(\mathbf{x})'\mathbf{v}\right] = \frac{1}{m}\mathbf{v}'\mathbf{X}'\mathbf{X}\mathbf{v}$$

Variance of projections

- Now suppose $\mathbf{v}$ has the dual representation $\mathbf{v} = \mathbf{X}'\alpha$. Average is given by

$$\mu_{\mathbf{v}} = \frac{1}{m}\alpha'\mathbf{X}\mathbf{X}'\mathbf{j} = \frac{1}{m}\alpha'\mathbf{K}\mathbf{j}$$

- average squared is given by

$$\frac{1}{m}\mathbf{v}'\mathbf{X}'\mathbf{X}\mathbf{v} = \frac{1}{m}\alpha'\mathbf{X}\mathbf{X}'\mathbf{X}\mathbf{X}'\alpha = \frac{1}{m}\alpha'\mathbf{K}^2\alpha$$

- Hence, variance in direction $\mathbf{v}$ is given by

$$\sigma_{\mathbf{v}}^2 = \frac{1}{m}\alpha'\mathbf{K}^2\alpha - \frac{1}{m^2}(\alpha'\mathbf{K}\mathbf{j})^2$$

Fisher discriminant

- The Fisher discriminant is a thresholded linear classifier:

$$f(\mathbf{x}) = \mathrm{sgn}(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b$$

where $\mathbf{w}$ is chosen to maximise the quotient:

$$J(\mathbf{w}) = \frac{(\mu_{\mathbf{w}}^{+} - \mu_{\mathbf{w}}^{-})^2}{(\sigma_{\mathbf{w}}^{+})^2 + (\sigma_{\mathbf{w}}^{-})^2}$$

- As with Ridge regression it makes sense to regularise if we are working in high-dimensional kernel spaces, so maximise

$$J(\mathbf{w}) = \frac{(\mu_{\mathbf{w}}^{+} - \mu_{\mathbf{w}}^{-})^2}{(\sigma_{\mathbf{w}}^{+})^2 + (\sigma_{\mathbf{w}}^{-})^2 + \lambda \|\mathbf{w}\|^2}$$

Fisher discriminant

- Using the results we now have we can substitute dual expressions for all of these quantities and solve using lagrange multipliers.

- The resulting classifier has dual variables

$$\alpha = (\mathbf{BK} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

where $\mathbf{B} = \mathbf{D} - \mathbf{C}$ with

$$\mathbf{C}_{ij} = \begin{cases} 2m^-/(mm^+) & \text{if } y_i = 1 = y_j \\ 2m^+/(mm^-) & \text{if } y_i = -1 = y_j \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{D} = \begin{cases} 2m^-/m & \text{if } i = j \text{ and } y_i = 1 \\ 2m^+/m & \text{if } i = j \text{ and } y_i = -1 \\ 0 & \text{otherwise} \end{cases}$$

and $b = 0.5\alpha\mathbf{K}\mathbf{t}$ with

$$\mathbf{t}_i = \begin{cases} 1/m^+ & \text{if } y_i = 1 \\ 1/m^- & \text{if } y_i = -1 \\ 0 & \text{otherwise} \end{cases}$$

giving a decision function

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) - b\right)$$

Preprocessing

- Corresponds to feature selection, or learning the feature space
- Note that in kernel methods the feature space is only determined up to orthogonal transformations (change of basis):

$$\hat{\phi}(\mathbf{x}) = \mathbf{U}\phi(\mathbf{x})$$

  for some orthogonal transformation $\mathbf{U}$ ($\mathbf{U}'\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}'$), then

$$\hat{\kappa}(\mathbf{x},\mathbf{z}) = \langle \mathbf{U}\phi(\mathbf{x}), \mathbf{U}\phi(\mathbf{z})\rangle = \phi(\mathbf{x})'\mathbf{U}'\mathbf{U}\phi(\mathbf{z}) = \phi(\mathbf{x})'\phi(\mathbf{z}) = \kappa(\mathbf{x},\mathbf{z})$$

- so feature selection in a kernel defined feature space is eqivalent to subspace projection

Subspace methods

- Principal components analysis: choose directions to maximise variance in the training data
- Canonical correlation analysis: choose directions to maximise correlations between two different views of the same objects
- Gram-Schmidt: greedily choose directions according to largest residual norms (not covered)
- Partial least squares: greedily choose directions with maximal covariance with the target (not covered)

In all cases we need kernel versions in order to apply these methods in high-dimensional kernel defined feature spaces

Principal Components Analysis

- PCA is a subspace method – that is it involves projecting the data into a lower dimensional space.

- Subspace is chosen to ensure maximal variance of the projections:

$$\mathbf{w} = \mathrm{argmax}_{\mathbf{w}:\|\mathbf{w}\|=1} \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w}$$

- This is equivalent to maximising the Raleigh quotient:

$$\frac{\mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w}}{\mathbf{w}'\mathbf{w}}$$

Principal Components Analysis

- We can optimise using Lagrange multipliers in order to remove the contraints:

  $$L(\mathbf{w}, \lambda) = \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} - \lambda \mathbf{w}'\mathbf{w}$$

  taking derivatives wrt $\mathbf{w}$ and setting equal to 0 gives:

  $$\mathbf{X}'\mathbf{X}\mathbf{w} = \lambda \mathbf{w}$$

  implying $\mathbf{w}$ is an eigenvector of $\mathbf{X}'\mathbf{X}$.

- Note that

  $$\lambda = \mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} = \sum_{i=1}^{m} \langle \mathbf{w}, \mathbf{x}_i \rangle^2$$

Principal Components Analysis

- So principal components analysis performs an eigenvalue decomposition of $\mathbf{X}'\mathbf{X}$ and projects into the space spanned by the first k eigenvectors

- Captures a total of

$$\sum_{i=1}^{k} \lambda_i$$

  of the overall variance:

$$\sum_{i=1}^{m} \|\mathbf{x}_i\|^2 = \sum_{i=1}^{n} \lambda_i = \mathrm{tr}(\mathbf{K})$$

Kernel PCA

- We would like to find a dual representation of the principal eigenvectors and hence of the projection function.

- Suppose that $\mathbf{w}, \lambda \neq 0$ is an eigenvector/eigenvalue pair for $\mathbf{X}'\mathbf{X}$, then $\mathbf{X}\mathbf{w}, \lambda$ is for $\mathbf{X}\mathbf{X}'$:

$$(\mathbf{X}\mathbf{X}')\mathbf{X}\mathbf{w} = \mathbf{X}(\mathbf{X}'\mathbf{X})\mathbf{w} = \lambda\mathbf{X}\mathbf{w}$$

- and vice versa $\alpha, \lambda \rightarrow \mathbf{X}'\alpha, \lambda$

$$(\mathbf{X}'\mathbf{X})\mathbf{X}'\alpha = \mathbf{X}'(\mathbf{X}\mathbf{X}')\alpha = \lambda\mathbf{X}'\alpha$$

- Note that we get back to where we started if we do it twice.

Kernel PCA

- Hence, 1-1 correspondence between eigenvectors corresponding to non-zero eigenvalues, but note that if $\|\alpha\| = 1$

$$\|\mathbf{X}'\alpha\|^2 = \alpha'\mathbf{X}\mathbf{X}'\alpha = \alpha'\mathbf{K}\alpha = \lambda$$

so if $\alpha^i, \lambda_i, i = 1, \ldots, k$ are first k eigenvectors/values of $\mathbf{K}$

$$\frac{1}{\sqrt{\lambda_i}}\alpha^i$$

are dual representations of first k eigenvectors $\mathbf{w}^1, \ldots, \mathbf{w}^k$ of $\mathbf{X}'\mathbf{X}$ with same eigenvalues.

- Computing projections:

$$\langle \mathbf{w}^i, \phi(\mathbf{x}) \rangle = \frac{1}{\sqrt{\lambda_i}}\langle \mathbf{X}'\alpha^i, \phi(\mathbf{x}) \rangle = \frac{1}{\sqrt{\lambda_i}}\sum_{j=1}^{m}\alpha_j^i\kappa(\mathbf{x}_i, \mathbf{x})$$

Kernel CCA

- Canonical correlation analysis finds correlations between different views of the same object:

$$\mathbf{x} \longmapsto (\phi_a(\mathbf{x}), \phi_b(\mathbf{x}))$$

- Examples:
  - documents in two languages
  - image and its caption
  - brain scan and corresponding activity description

- Seek $\mathbf{w}_a$ creating feature $x_a = \mathbf{w}_a' \phi_a(\mathbf{x})$ and $\mathbf{w}_b$ creating feature $x_b = \mathbf{w}_b' \phi_b(\mathbf{x})$ that maximise:

$$\rho = \frac{\hat{\mathbb{E}}[x_a x_b]}{\sqrt{\hat{\mathbb{E}}[x_a^2]\hat{\mathbb{E}}[x_b^2]}} = \frac{\hat{\mathbb{E}}[\mathbf{w}_a' \phi_a(\mathbf{x}) \phi_b(\mathbf{x})' \mathbf{w}_b]}{\sqrt{\hat{\mathbb{E}}[(\mathbf{w}_a' \phi_a(\mathbf{x}))^2]\hat{\mathbb{E}}[(\mathbf{w}_a' \phi_a(\mathbf{x}))^2]}}$$

Kernel CCA

- Using our standard notation

$$\rho = \frac{\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_b\mathbf{w}_b}{\sqrt{\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_a\mathbf{w}_a\mathbf{w}_b'\mathbf{X}_b'\mathbf{X}_b\mathbf{w}_b}}$$

- Since invariant to rescalings we can set two factors in denominator equal to 1. Using Lagrange multipliers obtain $L(\mathbf{w}_a, \mathbf{w}_b, \lambda_a, \lambda_b)$ as

$$\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_b\mathbf{w}_b - \frac{\lambda_a}{2}\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_a\mathbf{w}_a - \frac{\lambda_b}{2}\mathbf{w}_b'\mathbf{X}_b'\mathbf{X}_b\mathbf{w}_b$$

- Gives coupled equations

$\mathbf{X}_a'\mathbf{X}_b\mathbf{w}_b = \lambda_a\mathbf{X}_a'\mathbf{X}_a\mathbf{w}_a$ and $\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_b = \lambda_b\mathbf{w}_b'\mathbf{X}_b'\mathbf{X}_b$

Kernel CCA

- Simple to verify that $\lambda_a = \lambda_b$
- Resulting in generalised eigenvalue problem:

$$\begin{pmatrix} \mathbf{0} & \mathbf{X}_a'\mathbf{X}_b \\ \mathbf{X}_b'\mathbf{X}_a & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{w}_a \\ \mathbf{w}_b \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{X}_a'\mathbf{X}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{X}_b'\mathbf{X}_b \end{pmatrix} \begin{pmatrix} \mathbf{w}_a \\ \mathbf{w}_b \end{pmatrix}$$

- We would like to make a kernel version of this procedure – but must ensure that the flexibility is controlled to rule out spurious correlations:

$$\rho = \max_{\mathbf{w}_a, \mathbf{w}_b} \mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_b\mathbf{w}_b$$

subject to: $(1-\tau)\mathbf{w}_a'\mathbf{X}_a'\mathbf{X}_a\mathbf{w}_a + \tau\mathbf{w}_a'\mathbf{w}_a = 1$

and $(1-\tau)\mathbf{w}_b'\mathbf{X}_b'\mathbf{X}_b\mathbf{w}_b + \tau\mathbf{w}_b'\mathbf{w}_b = 1$

Kernel CCA

- We now dualise by letting $\mathbf{w}_a = \mathbf{X}_a'\alpha$ and $\mathbf{w}_b = \mathbf{X}_b'\beta$ to obtain:

$$\begin{array}{rcl} \rho & = & \max_{\alpha,\beta} \alpha'\mathbf{K}_a\mathbf{K}_b\beta \\ \text{subject to:} & & (1-\tau)\alpha'\mathbf{K}_a^2\alpha + \tau\alpha'\mathbf{K}_a\alpha = 1 \\ \text{and} & & (1-\tau)\beta'\mathbf{K}_b^2\beta + \tau\beta'\mathbf{K}_b\beta = 1 \end{array}$$

- giving the generalised eigenvalue problem

$$\begin{pmatrix} \mathbf{0} & \mathbf{K}_a\mathbf{K}_b \\ \mathbf{K}_b\mathbf{K}_a & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda \begin{pmatrix} (1-\tau)\,\mathbf{K}_a^2 + \tau\mathbf{K}_a & \mathbf{0} \\ \mathbf{0} & (1-\tau)\,\mathbf{K}_b^2 + \tau\mathbf{K}_b \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Part 3

- Statistical analysis of the stability of patterns.
- Rademacher complexity.
- Generalisation of SVMs
- Support Vector Machine Optimisation

Generalisation of a learner

- Assume that we have a learning algorithm $\mathcal{A}$ that chooses a function $\mathcal{A}_{\mathcal{F}}(S)$ from a function space $\mathcal{F}$ in response to the training set S.

- From a statistical point of view the quantity of interest is the random variable:

$$\epsilon(S, \mathcal{A}, \mathcal{F}) = \mathbb{E}_{(\mathbf{x},y)}\left[\ell(\mathcal{A}_{\mathcal{F}}(S), \mathbf{x}, y)\right],$$

  where ℓ is a 'loss' function that measures the discrepancy between $\mathcal{A}_{\mathcal{F}}(S)(\mathbf{x})$ and y.

Generalisation of a learner

- For example, in the case of classification ℓ is 1 if the two disagree and 0 otherwise, while for regression it could be the square of the difference between $\mathcal{A}_{\mathcal{F}}(S)(\mathbf{x})$ and y.

- We refer to the random variable $\epsilon(S, \mathcal{A}, \mathcal{F})$ as the generalisation of the learner.

Example of Generalisation I

- We consider the Breast Cancer dataset from the UCI repository.
- Use the simple Parzen window classifier described in Part 2: weight vector is

$$\mathbf{w}^+ - \mathbf{w}^-$$

where $\mathbf{w}^+$ is the average of the positive training examples and $\mathbf{w}^-$ is average of negative training examples. Threshold is set so hyperplane bisects the line joining these two points.

Example of Generalisation II

- Given a size m of the training set, by repeatedly drawing random training sets S we estimate the distribution of

$$\epsilon(S, \mathcal{A}, \mathcal{F}) = \mathbb{E}_{(\mathbf{x},y)} \left[\ell(\mathcal{A}_{\mathcal{F}}(S), \mathbf{x}, y)\right],$$

by using the test set error as a proxy for the true generalisation.

- We plot the histogram and the average of the distribution for various sizes of training set – initially the whole dataset gives a single value if we use training and test as the all the examples, but then we plot for training set sizes:

$$342, 273, 205, 137, 68, 34, 27, 20, 14, 7.$$

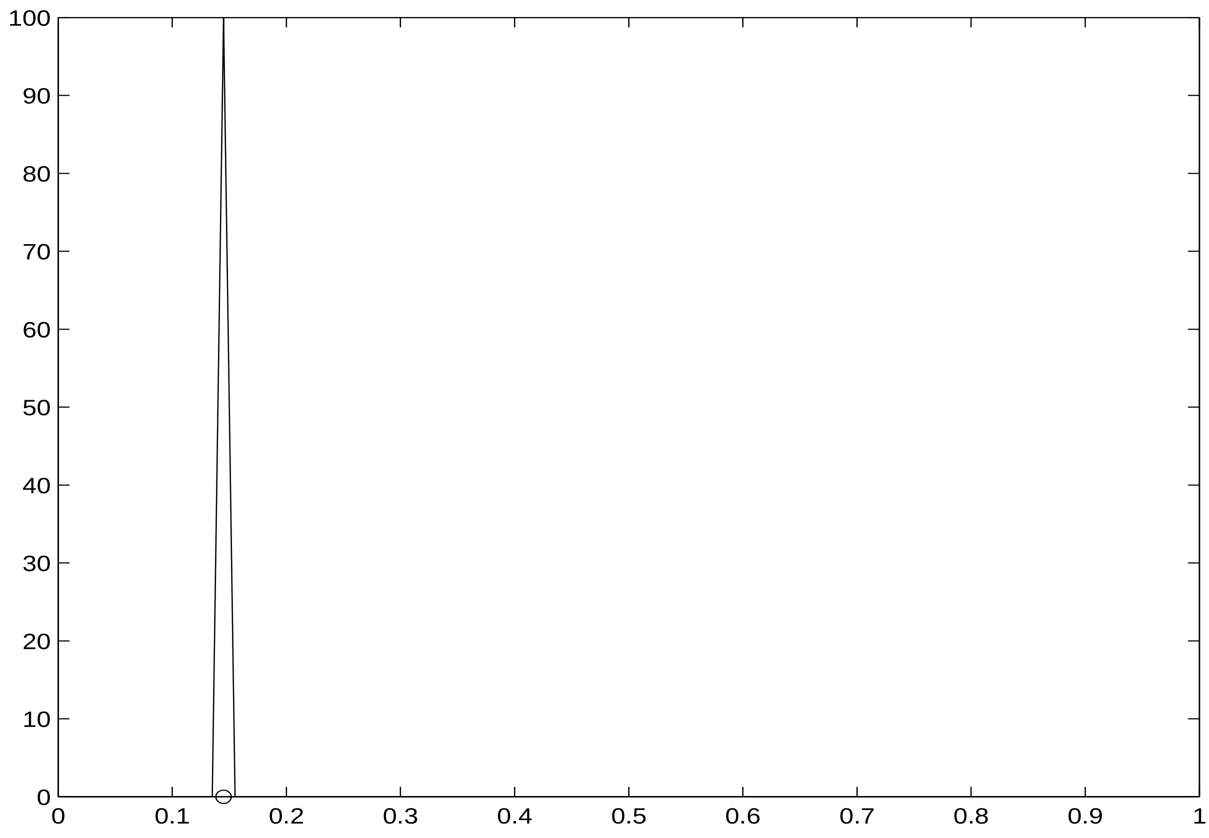
Example of Generalisation III

- Since the expected classifier is in all cases the same:

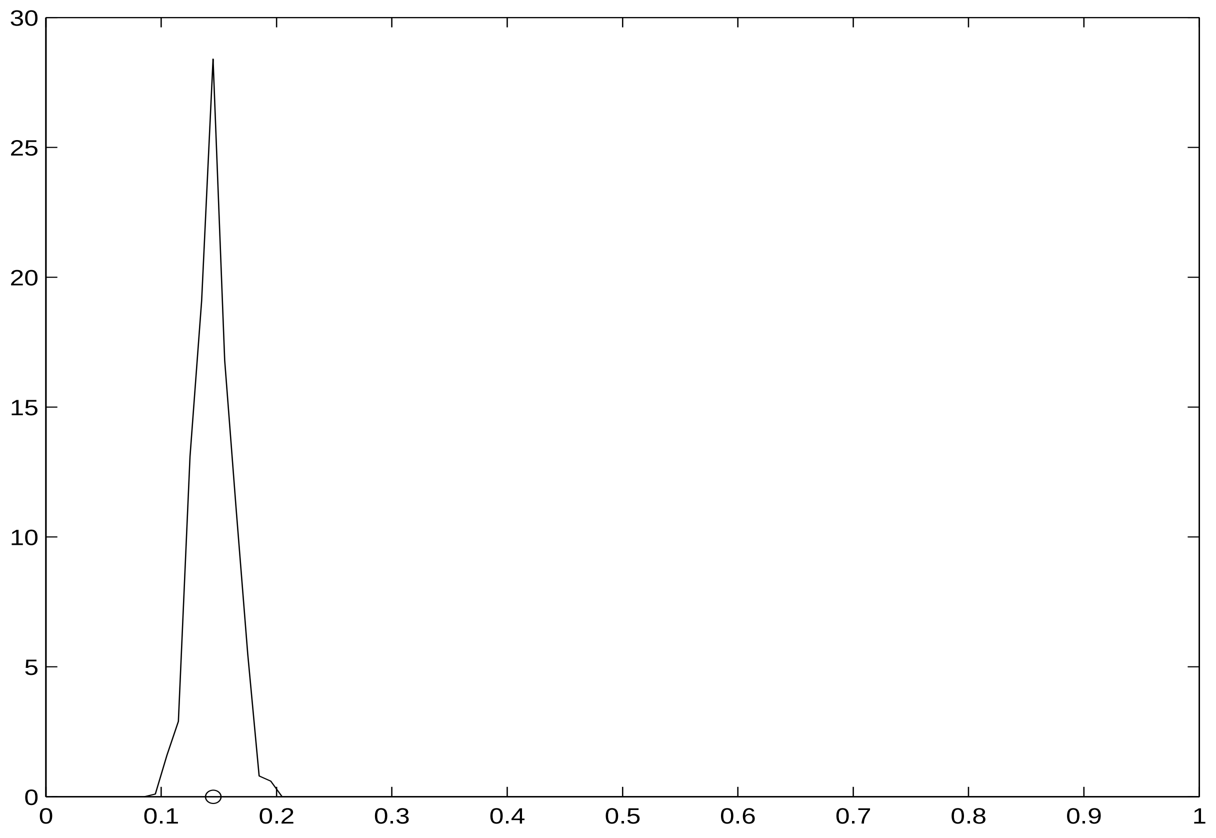
$$\begin{aligned}\mathbb{E}\left[\mathcal{A}_{\mathcal{F}}(S)\right] &= \mathbb{E}_S\left[\mathbf{w}_S^+ - \mathbf{w}_S^-\right] \\ &= \mathbb{E}_S\left[\mathbf{w}_S^+\right] - \mathbb{E}_S\left[\mathbf{w}_S^-\right] \\ &= \mathbb{E}_{y=+1}\left[\mathbf{x}\right] - \mathbb{E}_{y=-1}\left[\mathbf{x}\right],\end{aligned}$$

  we do not expect large differences in the average of the distribution, though the non-linearity of the loss function means they won't be the same exactly.

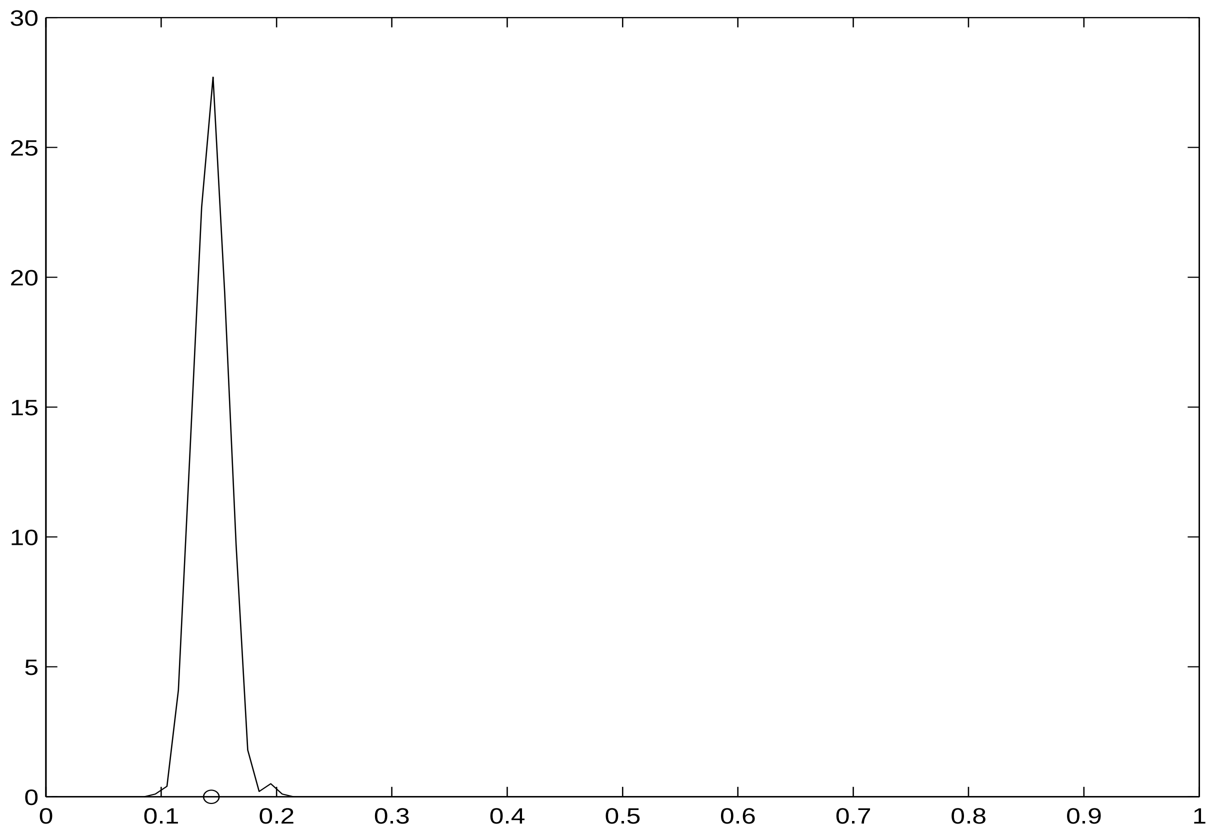
Error distribution: full dataset



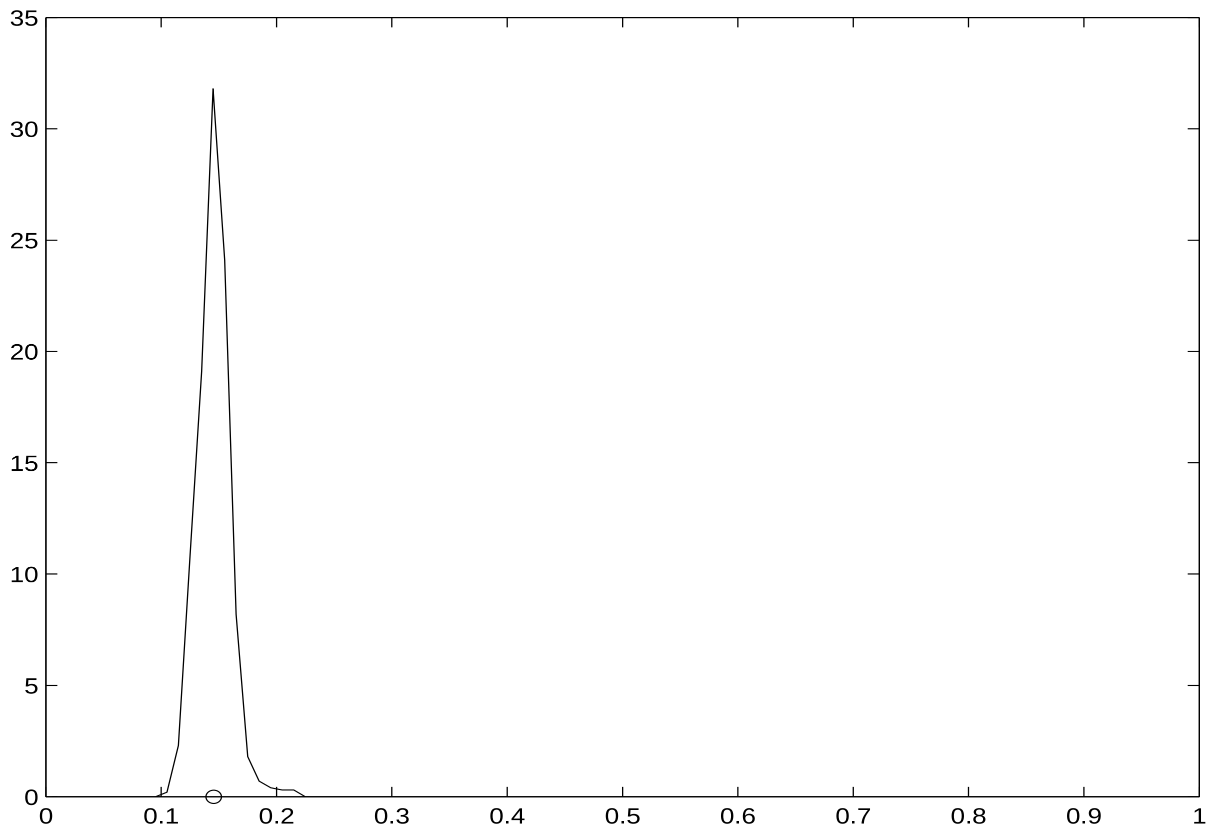
Error distribution: dataset size: 342



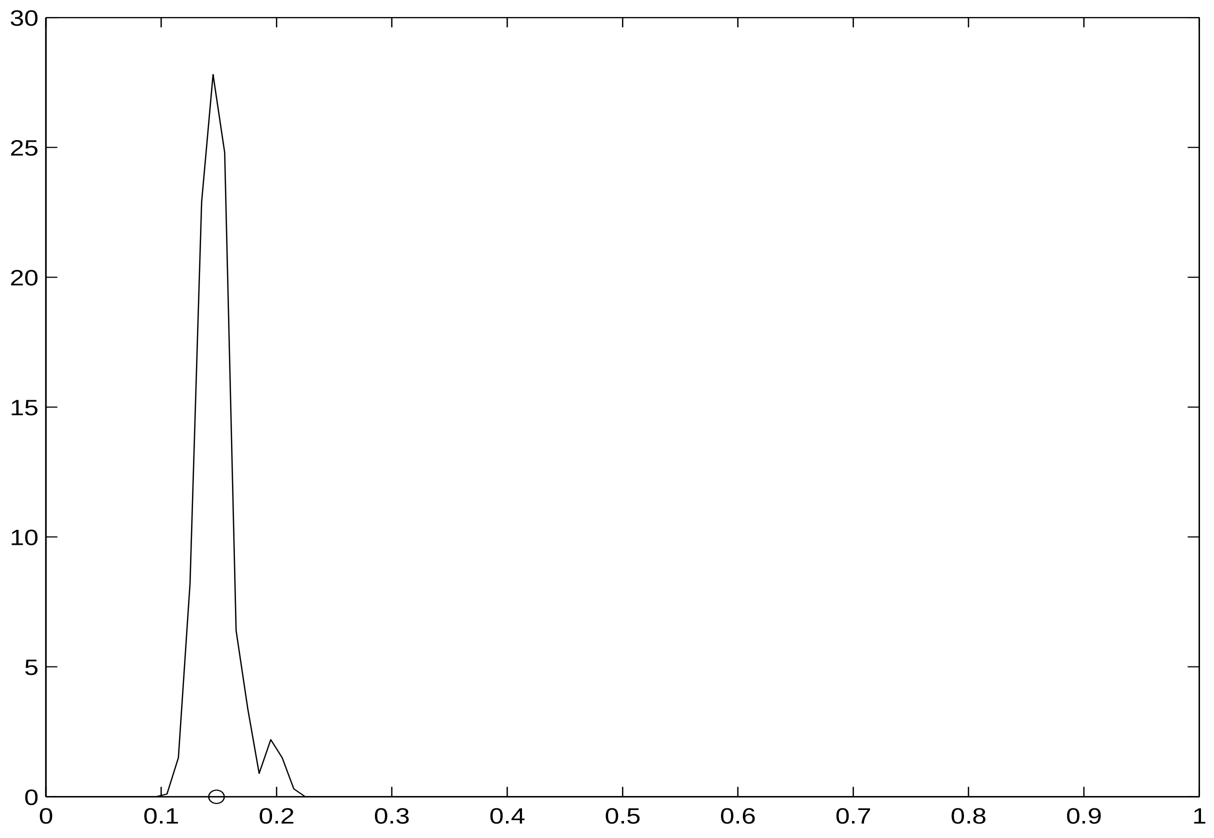
Error distribution: dataset size: 273



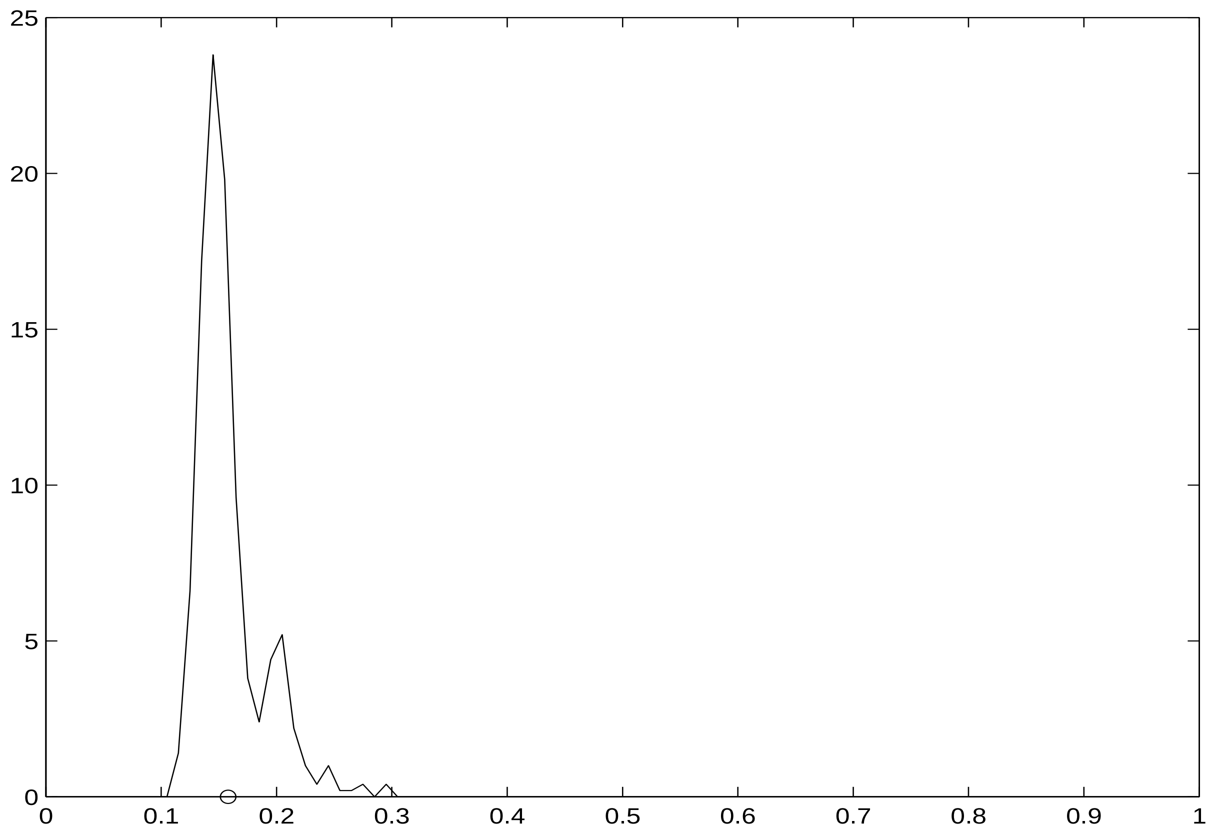
Error distribution: dataset size: 205



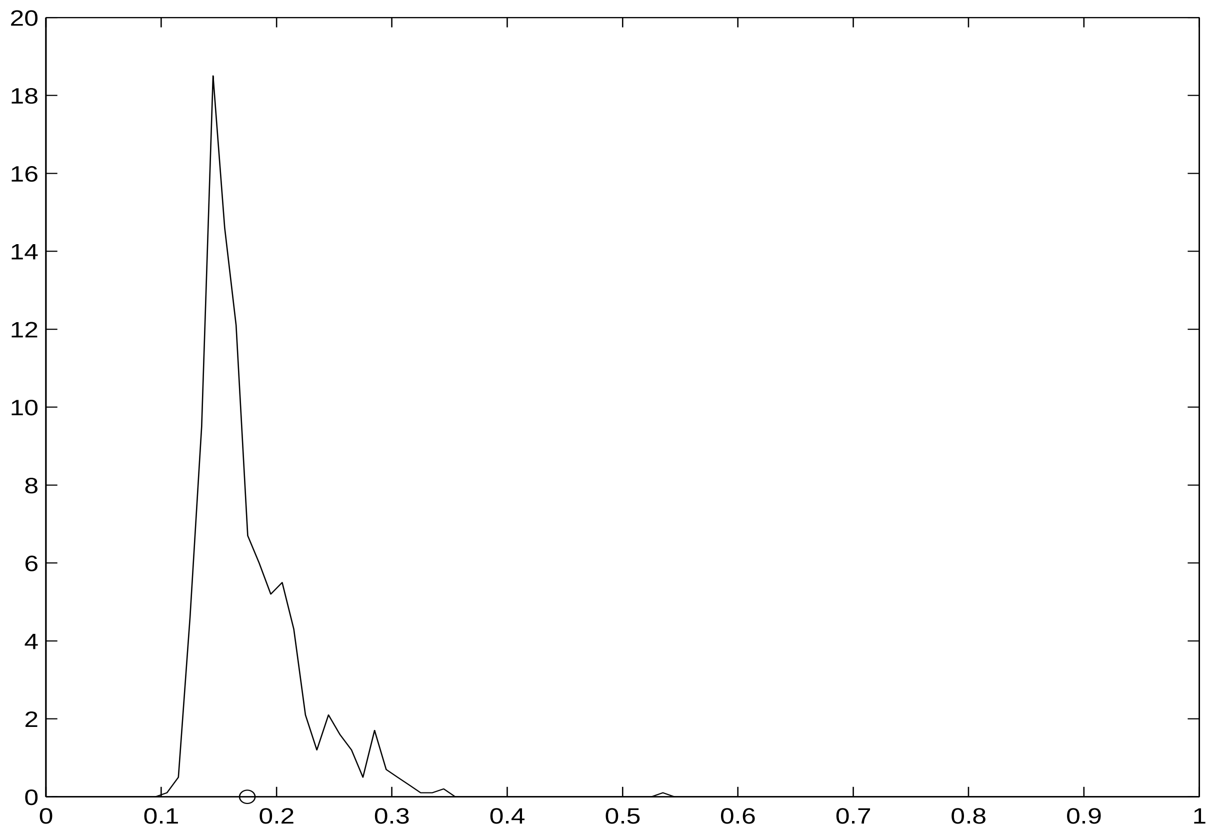
Error distribution: dataset size: 137



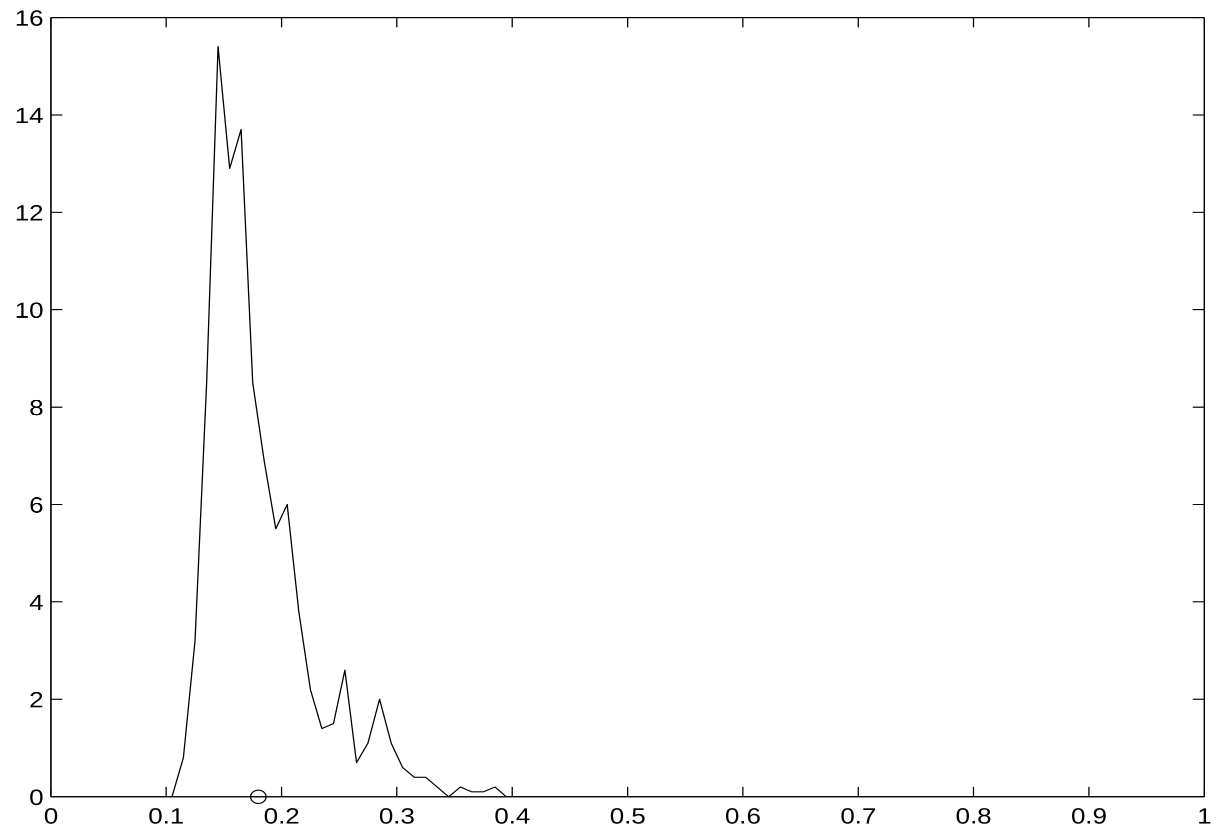
Error distribution: dataset size: 68



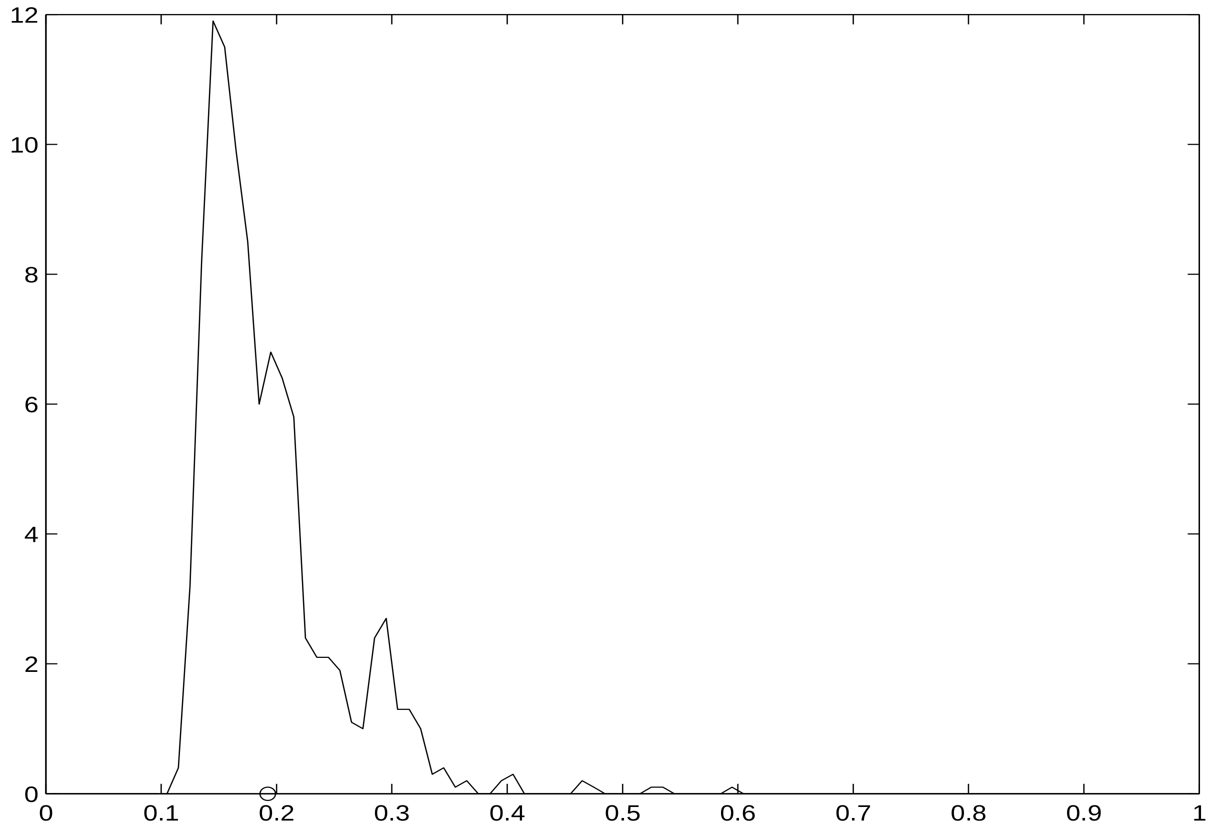
Error distribution: dataset size: 34



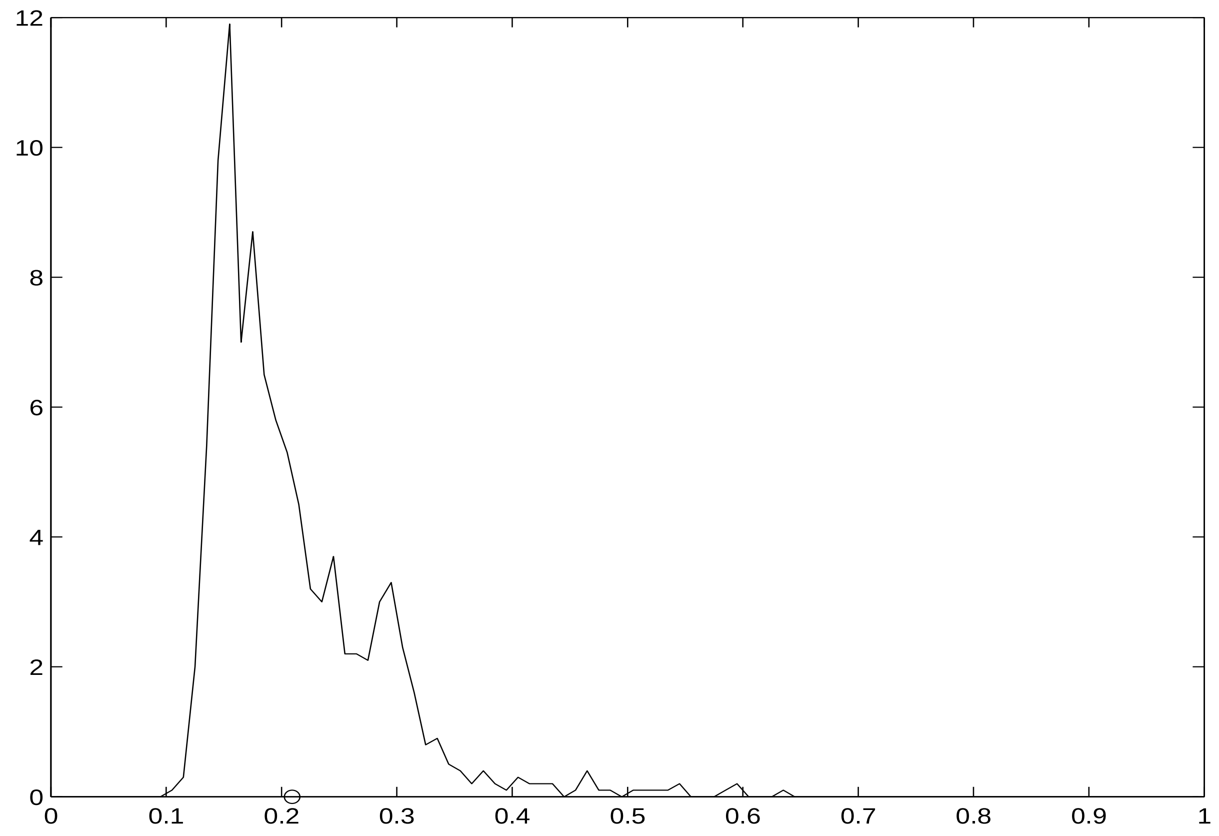
Error distribution: dataset size: 27



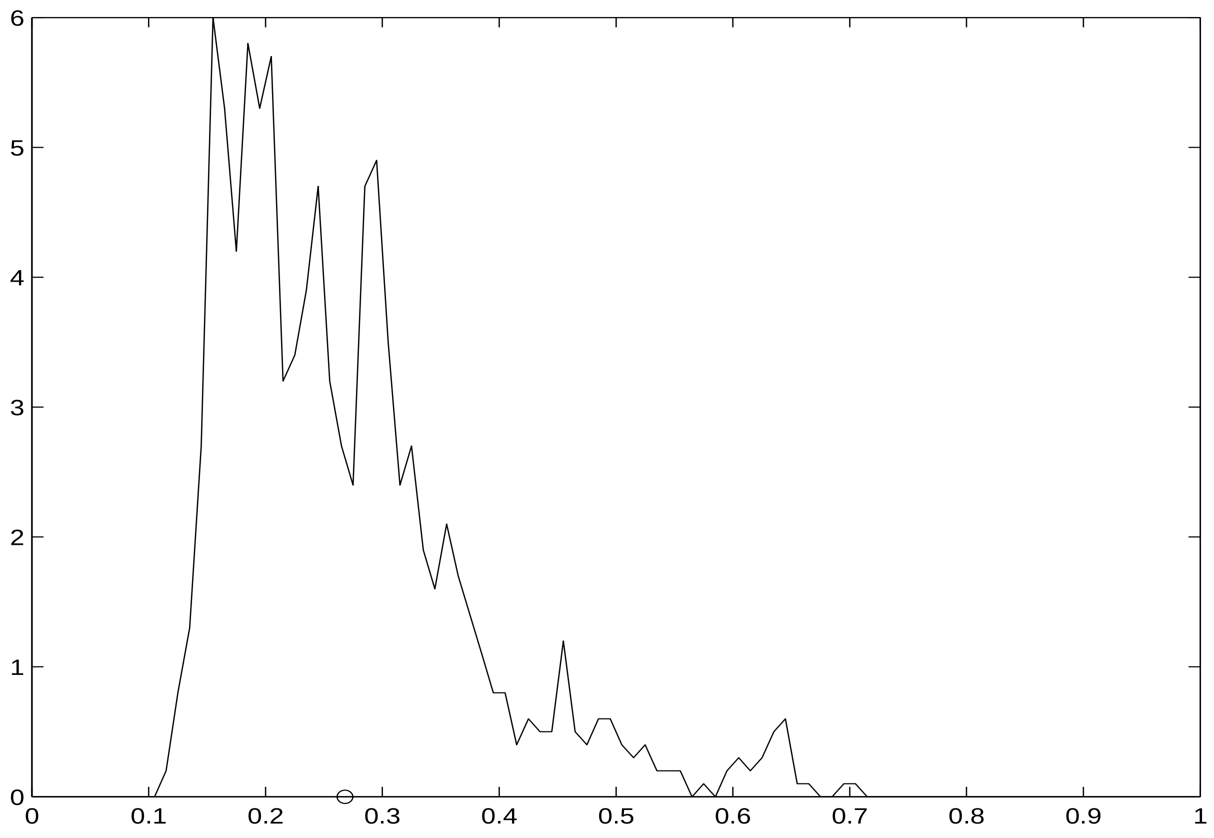
Error distribution: dataset size: 20



Error distribution: dataset size: 14



Error distribution: dataset size: 7



Observations

- Things can get bad if number of training examples small compared to dimension (in this case input dimension is 9)
- Mean can be bad predictor of true generalisation i.e. things can look okay in expectation, but still go badly wrong
- Key ingredient of learning keep flexibility high while still ensuring good generalisation

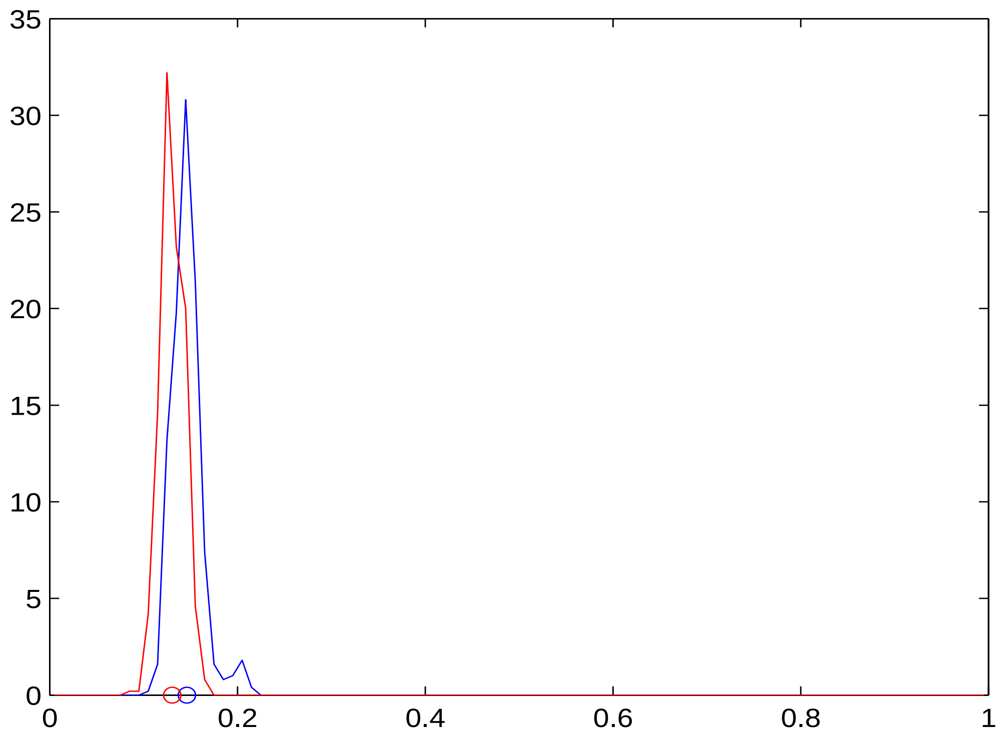
Controlling generalisation

- The critical method of controlling generalisation for classification is to force a large margin on the training data
- Equivalent to minimising the norm while keeping the separation fixed (at say ± 1)
- Support Vector Machines implement this strategy

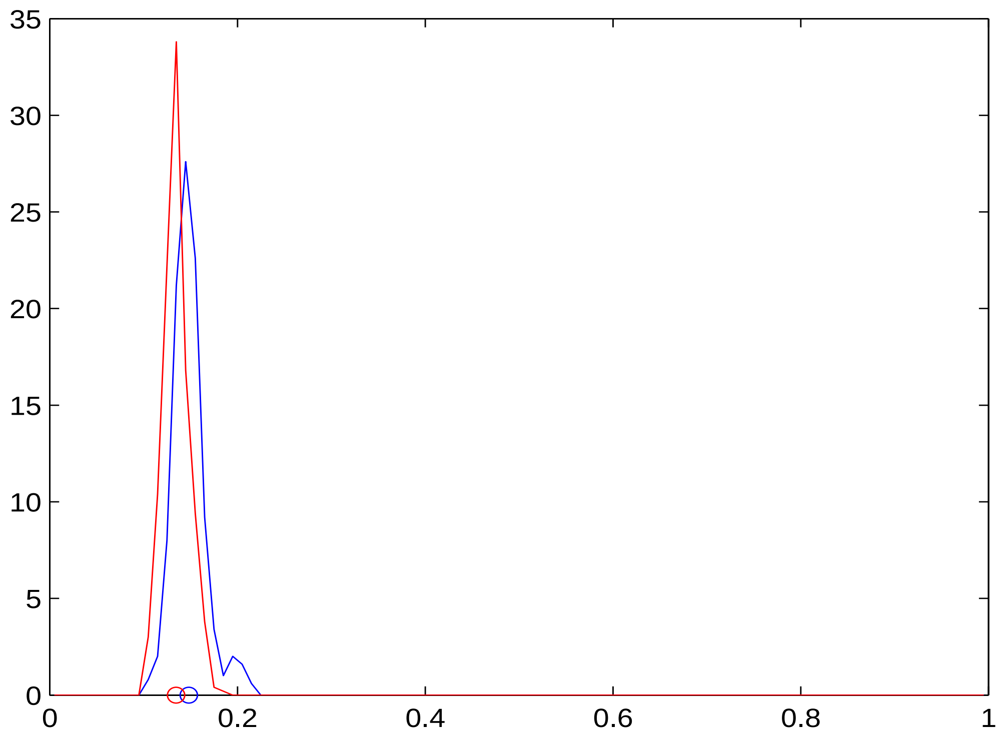
Controlling generalisation

- Now consider using an SVM on the same data and compare the distribution of generalisations
- SVM distribution in red

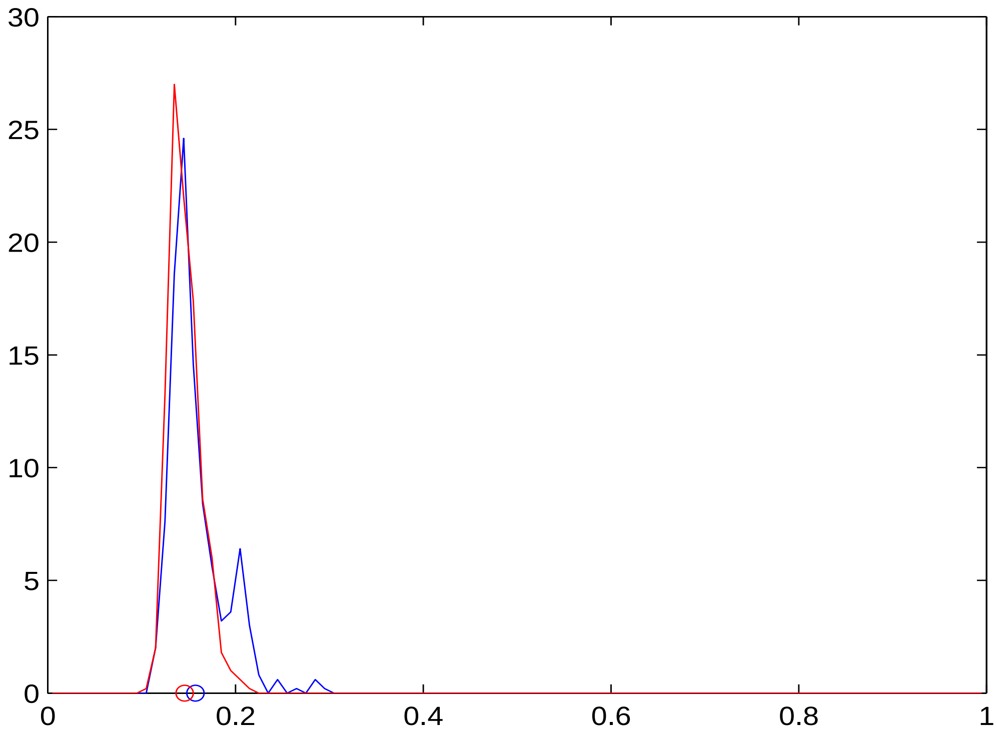
Error distribution: dataset size: 205



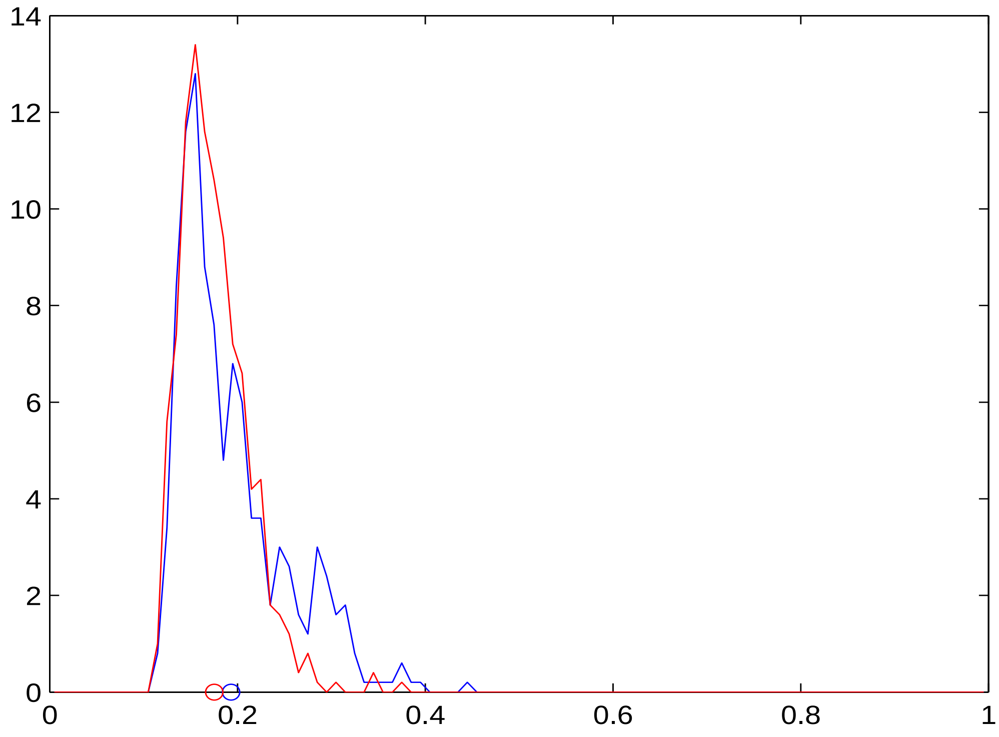
Error distribution: dataset size: 137



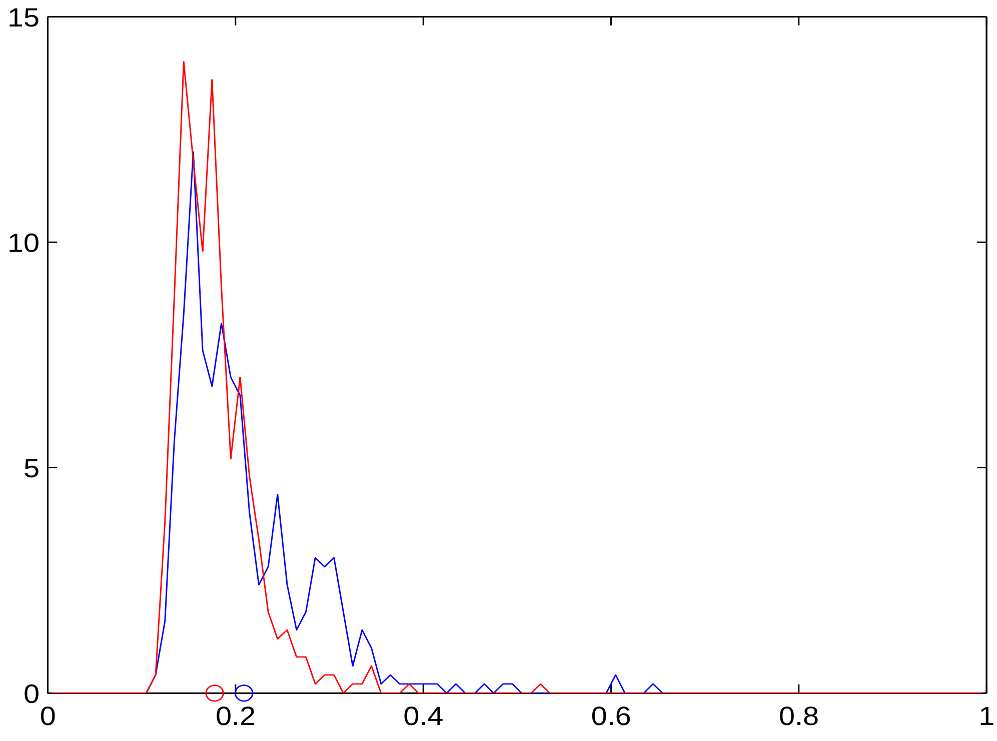
Error distribution: dataset size: 68



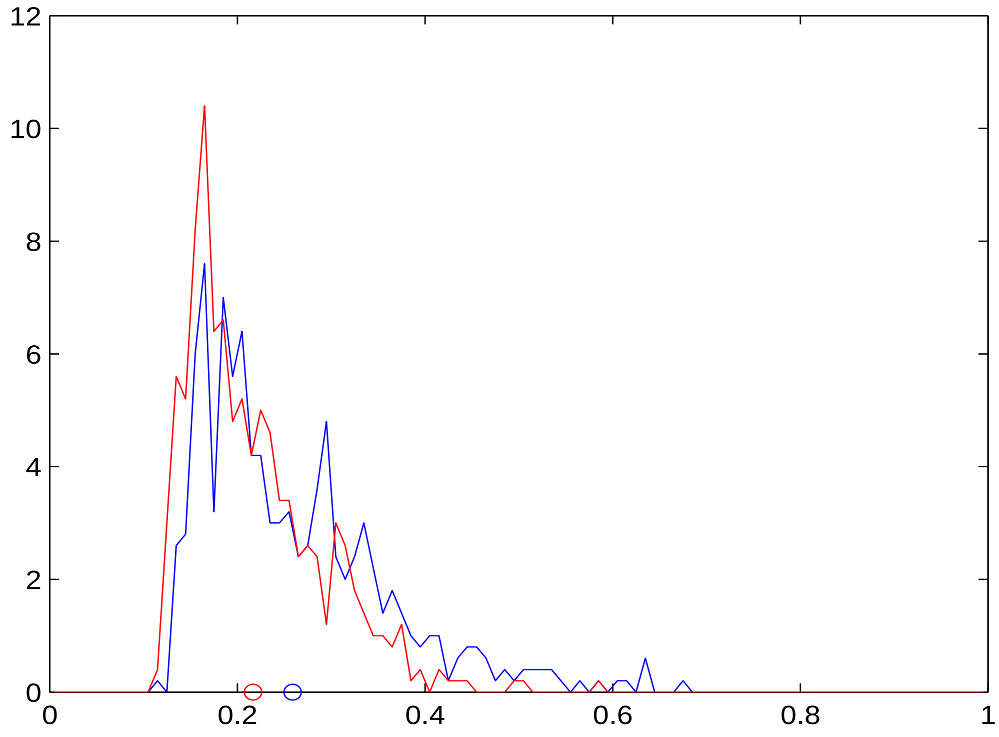
Error distribution: dataset size: 20



Error distribution: dataset size: 14



Error distribution: dataset size: 7



Expected versus confident bounds

- For a finite sample the generalisation $\epsilon(S, \mathcal{A}, \mathcal{F})$ has a distribution depending on the algorithm, function class and sample size m.

- Traditional statistics as indicated above has concentrated on the mean of this distribution – but this quantity can be misleading, eg for low fold cross-validation.

Expected versus confident bounds cont.

- Statistical learning theory has preferred to analyse the tail of the distribution, finding a bound which holds with high probability.

- This looks like a statistical test – significant at a 1% confidence means that the chances of the conclusion not being true are less than 1% over random samples of that size.

- This is also the source of the acronym PAC: probably approximately correct, the ‘confidence’ parameter δ is the probability that we have been misled by the training set.

Concentration inequalities

- Statistical Learning is concerned with the reliability or stability of inferences made from a random sample.
- Random variables with this property have been a subject of ongoing interest to probabilists and statisticians.

Concentration inequalities cont.

- As an example consider the mean of a sample of m 1-dimensional random variables $X_1, \ldots, X_m$:

$$S_m = \frac{1}{m} \sum_{i=1}^{m} X_i.$$

- Hoeffding's inequality states that if $X_i \in [a_i, b_i]$

$$P\{|S_m - \mathbb{E}[S_m]| \geq \epsilon\} \leq 2 \exp\left(-\frac{2m^2\epsilon^2}{\sum_{i=1}^{m}(b_i - a_i)^2} \right)$$

  Note how the probability falls off exponentially with the distance from the mean and with the number of variables.

Concentration for SLT

- We are now going to look at deriving SLT results from concentration inequalities.

- Perhaps the best known form is due to McDiarmid (although he was actually representing previously derived results):

McDiarmid's inequality

Theorem 1. *Let $X_1, \ldots, X_n$ be independent random variables taking values in a set A, and assume that $f : A^n \to \mathbb{R}$ satisfies*

$$\sup_{x_1,\ldots,x_n,\hat{x}_i \in A} |f(x_1, \ldots, x_n) - f(x_1, \ldots, \hat{x}_i, x_{i+1}, \ldots, x_n)| \leq c_i,$$

for $1 \leq i \leq n$. Then for all $\epsilon > 0$,

$$P\left\{f\left(X_1, \ldots, X_n\right) - \mathbb{E}f\left(X_1, \ldots, X_n\right) \geq \epsilon\right\} \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

- Hoeffding is a special case when $f(x_1, \ldots, x_n) = S_n$

Using McDiarmid

- By setting the right hand side equal to δ, we can always invert McDiarmid to get a high confidence bound: with probability at least $1 - \delta$

$$f\left(X_1, \ldots, X_n\right) < \mathbb{E} f\left(X_1, \ldots, X_n\right) + \sqrt{\frac{\sum_{i=1}^{n} c_i^2}{2} \log \frac{1}{\delta}}$$

- If $c_i = c/n$ for each i this reduces to

$$f\left(X_1, \ldots, X_n\right) < \mathbb{E} f\left(X_1, \ldots, X_n\right) + \sqrt{\frac{c^2}{2n} \log \frac{1}{\delta}}$$

Rademacher complexity

- Rademacher complexity is a new way of measuring the complexity of a function class. It arises naturally if we rerun the proof using the double sample trick and symmetrisation but look at what is actually needed to continue the proof:

Rademacher proof beginnings

For a fixed $f \in \mathcal{F}$ we have

$$\mathbb{E}\left[f(\mathbf{z})\right] \leq \hat{\mathbb{E}}\left[f(\mathbf{z})\right] + \sup_{h \in \mathcal{F}} \left(\mathbb{E}[h] - \hat{\mathbb{E}}[h]\right).$$

where $\mathcal{F}$ is a class of functions mapping from Z to $[0,1]$ and $\hat{\mathbb{E}}$ denotes the sample average.

We must bound the size of the second term. First apply McDiarmid's inequality to obtain ($c_i = 1/m$ for all i) with probability at least $1-\delta$:

$$\sup_{h \in \mathcal{F}} \left(\mathbb{E}[h] - \hat{\mathbb{E}}[h]\right) \leq \mathbb{E}_S \left[\sup_{h \in \mathcal{F}} \left(\mathbb{E}[h] - \hat{\mathbb{E}}[h]\right)\right] + \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

Deriving double sample result

- We can now move to the ghost sample by simply observing that $\mathbb{E}[h] = \mathbb{E}_{\tilde{S}}\left[\hat{\mathbb{E}}[h]\right]$:

$$\mathbb{E}_S\left[\sup_{h\in\mathcal{F}}\left(\mathbb{E}[h] - \hat{\mathbb{E}}[h]\right)\right] =$$

$$\mathbb{E}_S\left[\sup_{h\in\mathcal{F}}\mathbb{E}_{\tilde{S}}\left[\frac{1}{m}\sum_{i=1}^{m} h(\tilde{\mathbf{z}}_i) - \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{z}_i)\middle| S\right]\right]$$

Deriving double sample result cont.

Since the sup of an expectation is less than or equal to the expectation of the sup (we can make the choice to optimise for each $\tilde{S}$) we have

$$\mathbb{E}_S \left[\sup_{h \in \mathcal{F}} \left(\mathbb{E}[h] - \hat{\mathbb{E}}[h] \right) \right] \leq$$

$$\mathbb{E}_S \mathbb{E}_{\tilde{S}} \left[\sup_{h \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \left(h(\tilde{\mathbf{z}}_i) - h(\mathbf{z}_i) \right) \right]$$

Adding symmetrisation

Here symmetrisation is again just swapping corresponding elements – but we can write this as multiplication by a variable σ_i which takes values ± 1 with equal probability:

$$\begin{aligned}
\mathbb{E}_S\left[\sup_{h\in\mathcal{F}}\left(\mathbb{E}[h]-\hat{\mathbb{E}}[h]\right)\right] \leq & \\
\leq & \quad \mathbb{E}_{\sigma S\tilde{S}}\left[\sup_{h\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^{m}\sigma_i\left(h(\tilde{\mathbf{z}}_i)-h(\mathbf{z}_i)\right)\right] \\
\leq & \quad 2\mathbb{E}_{S\sigma}\left[\sup_{h\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^{m}\sigma_i h(\mathbf{z}_i)\right] \\
= & \quad R_m\left(\mathcal{F}\right),
\end{aligned}$$

assuming $\mathcal{F}$ closed under negation $f \mapsto -f$.

Rademacher complexity

The Rademacher complexity provides a way of measuring the complexity of a function class $\mathcal{F}$ by testing how well on average it can align with random noise:

$$R_m(\mathcal{F}) = \mathbb{E}_{S\sigma}\left[\sup_{f\in\mathcal{F}} \frac{2}{m}\sum_{i=1}^{m}\sigma_i f\left(\mathbf{z}_i\right)\right].$$

is known as the Rademacher complexity of the function class $\mathcal{F}$.

Main Rademacher theorem

The main theorem of Rademacher complexity: with probability at least $1 - \delta$ over random samples S of size m, every $f \in \mathcal{F}$ satisfies

$$\mathbb{E}\left[f(\mathbf{z})\right] \leq \hat{\mathbb{E}}\left[f(\mathbf{z})\right] + R_m(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{2m}}$$

- Note that Rademacher complexity gives the expected value of the maximal correlation with random noise – a very natural measure of capacity.

- Note that the Rademacher complexity is distribution dependent since it involves an expectation over the choice of sample – this might seem hard to compute.

Empirical Rademacher theorem

- Since the empirical Rademacher complexity

$$\hat{R}_m(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{2}{m} \sum_{i=1}^{m} \sigma_i f\left(\mathbf{z}_i\right) \middle| \mathbf{z}_1, \ldots, \mathbf{z}_m \right]$$

is concentrated, we can make a further application of McDiarmid to obtain with probability at least $1 - \delta$

$$\mathbb{E}_\mathcal{D}\left[f(\mathbf{z})\right] \leq \hat{\mathbb{E}}\left[f(\mathbf{z})\right] + \hat{R}_m(\mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}.$$

Application to large margin classification

- Rademacher complexity comes into its own for Boosting and SVMs.

Application to Boosting

- We can view Boosting as seeking a function from the class (H is the set of weak learners)

$$\left\{ \sum_{h \in H} a_h h(\mathbf{x}) : a_h \geq 0, \sum_{h \in H} a_h \leq B \right\} = \mathrm{conv}_B(H)$$

  by minimising some function of the margin distribution (assume H closed under negation).

- Adaboost corresponds to optimising an exponential function of the margin over this set of functions.

- We will see how to include the margin in the analysis later, but concentrate on computing the Rademacher complexity for now.

Rademacher complexity of convex hulls

Rademacher complexity has a very nice property for convex hull classes:

$$\begin{aligned}
\hat{R}_m(\mathrm{conv}_B(H)) &= \frac{2}{m}\mathbb{E}_\sigma\left[\sup_{h_j \in H, \sum_j a_j \leq B} \sum_{i=1}^{m} \sigma_i \sum_j a_j h_j(\mathbf{x}_i)\right] \\
&\leq \frac{2}{m}\mathbb{E}_\sigma\left[\sup_{h_j \in H, \sum_j a_j \leq B} \sum_j a_j \sum_{i=1}^{m} \sigma_i h_j(\mathbf{x}_i)\right] \\
&\leq \frac{2}{m}\mathbb{E}_\sigma\left[\sup_{h_j \in H} B \sum_{i=1}^{m} \sigma_i h_j(\mathbf{x}_i)\right] \\
&\leq B\hat{R}_m(H).
\end{aligned}$$

Rademacher complexity of convex hulls cont.

- Hence, we can move to the convex hull without incurring any complexity penalty for $B = 1$!

Rademacher complexity for SVMs

- The Rademacher complexity of a class of linear functions with bounded 2-norm:

$$\left\{\mathbf{x} \to \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \colon \alpha' \mathbf{K} \alpha \leq B^2 \right\} \subseteq$$
$$\subseteq \{\mathbf{x} \to \langle \mathbf{w}, \phi(\mathbf{x}) \rangle : \|\mathbf{w}\| \leq B\}$$
$$= \mathcal{F}_B,$$

where we assume a kernel defined feature space with

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \kappa(\mathbf{x}, \mathbf{z}).$$

Rademacher complexity of $\mathcal{F}_B$

The following derivation gives the result

$$\begin{aligned}
\hat{R}_m(\mathcal{F}_B) &= \mathbb{E}_\sigma\left[\sup_{f\in\mathcal{F}_B}\left|\frac{2}{m}\sum_{i=1}^{m}\sigma_i f\left(\mathbf{x}_i\right)\right|\right] \\
&= \mathbb{E}_\sigma\left[\sup_{\|\mathbf{w}\|\leq B}\left|\left\langle \mathbf{w}, \frac{2}{m}\sum_{i=1}^{m}\sigma_i \phi\left(\mathbf{x}_i\right)\right\rangle\right|\right] \\
&\leq \frac{2B}{m}\mathbb{E}_\sigma\left[\left\|\sum_{i=1}^{m}\sigma_i\phi(\mathbf{x}_i)\right\|\right] \\
&= \frac{2B}{m}\mathbb{E}_\sigma\left[\left(\left\langle\sum_{i=1}^{m}\sigma_i\phi(\mathbf{x}_i), \sum_{j=1}^{m}\sigma_j\phi(\mathbf{x}_j)\right\rangle\right)^{1/2}\right]
\end{aligned}$$

$$\leq \frac{2B}{m}\left(\mathbb{E}_\sigma\left[\sum_{i,j=1}^{m}\sigma_i\sigma_j\kappa(\mathbf{x}_i,\mathbf{x}_j)\right]\right)^{1/2} = \frac{2B}{m}\sqrt{\sum_{i=1}^{m}\kappa(\mathbf{x}_i,\mathbf{x}_i)}$$

Support Vector Machines (SVM)

- SVM seeks linear function in a feature space defined implicitly via a kernel κ:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

  that optimises a bound on the generalisation.

- The first step is to introduce a loss function which upper bounds the discrete loss

$$P(y \neq \mathrm{sgn}(g(\mathbf{x}))) = \mathbb{E}\left[\mathcal{H}(-yg(\mathbf{x}))\right],$$

  where $\mathcal{H}$ is the Heaviside function.

Margins in SVMs

- Critical to the bound will be the margin of the classifier

  $$\gamma(\mathbf{x}, y) = yg(\mathbf{x}) = y(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b) :$$

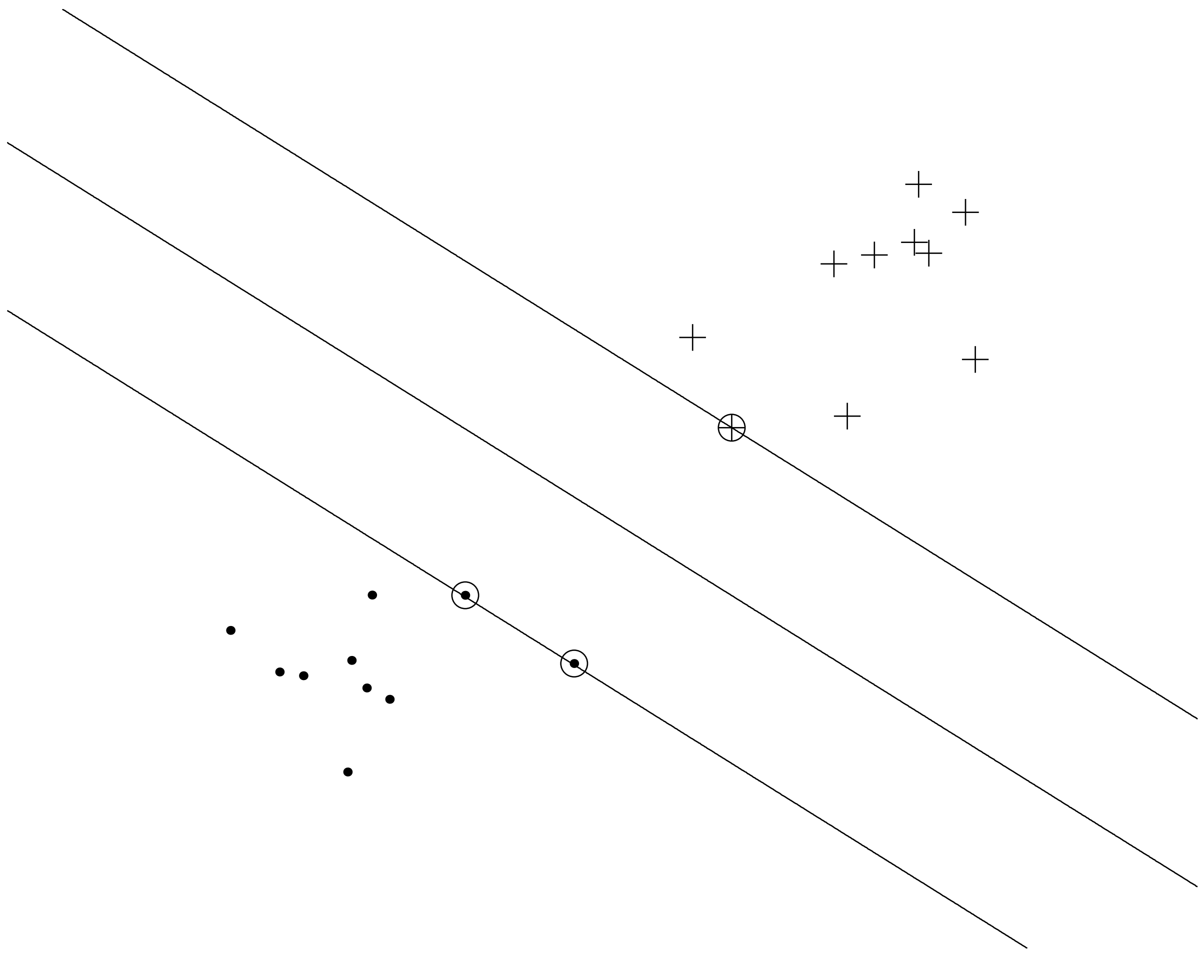
  positive if correctly classified, and measures distance from the separating hyperplane when the weight vector is normalised.

- The margin of a linear function g is

  $$\gamma(g) = \min_i \gamma(\mathbf{x}_i, y_i)$$

  though this is frequently increased to allow some 'margin errors'.

Margins in SVMs



Applying the Rademacher theorem

- Consider the loss function $\mathcal{A} : \mathbb{R} \to [0, 1]$, given by

$$\mathcal{A}(a) = \begin{cases} 1, & \text{if } a > 0; \\ 1 + a/\gamma, & \text{if } -\gamma \le a \le 0; \\ 0, & \text{otherwise.} \end{cases}$$

- By the Rademacher Theorem and since the loss function $\mathcal{A}$ dominates $\mathcal{H}$, we have that

$$\begin{aligned} \mathbb{E}\left[\mathcal{H}(-yg(\mathbf{x}))\right] &\le \mathbb{E}\left[\mathcal{A}(-yg(\mathbf{x}))\right] \\ &\le \hat{\mathbb{E}}\left[\mathcal{A}(-yg(\mathbf{x}))\right] + \\ &\quad \hat{R}_m(\mathcal{A} \circ \mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}. \end{aligned}$$

Empirical loss and slack variables

- But the function $\mathcal{A}(-y_i g(\mathbf{x}_i)) \leq \xi_i/\gamma$, for $i = 1, \ldots, m$, and so

$$\mathbb{E}\left[\mathcal{H}(-yg(\mathbf{x}))\right] \leq \frac{1}{m\gamma}\sum_{i=1}^{m}\xi_i + \hat{R}_m(\mathcal{A}\circ\mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}.$$

- The final missing ingredient to complete the bound is to bound $\hat{R}_m(\mathcal{A}\circ\mathcal{F})$ in terms of $\hat{R}_m(\mathcal{F})$.

- This can be obtained in terms of the maximal slope of the function $\mathcal{A}$: $\hat{R}_m(\mathcal{A}\circ\mathcal{F}) \leq \frac{2}{\gamma}\hat{R}_m(\mathcal{F})$.

Final SVM bound

- Assembling the result we obtain:

$$\begin{aligned} P(y \neq \operatorname{sgn}(g(\mathbf{x}))) &= \mathbb{E}\left[\mathcal{H}(-yg(\mathbf{x}))\right] \\ &\leq \frac{1}{m\gamma}\sum_{i=1}^{m}\xi_i + \frac{4}{m\gamma}\sqrt{\sum_{i=1}^{m}\kappa(\mathbf{x}_i,\mathbf{x}_i)} + 3\sqrt{\frac{\ln(2/\delta)}{2m}} \end{aligned}$$

- Note that for the Gaussian kernel this reduces to

$$P(y \neq \operatorname{sgn}(g(\mathbf{x}))) \leq \frac{1}{m\gamma}\sum_{i=1}^{m}\xi_i + \frac{4}{\sqrt{m}\gamma} + 3\sqrt{\frac{\ln(2/\delta)}{2m}}$$

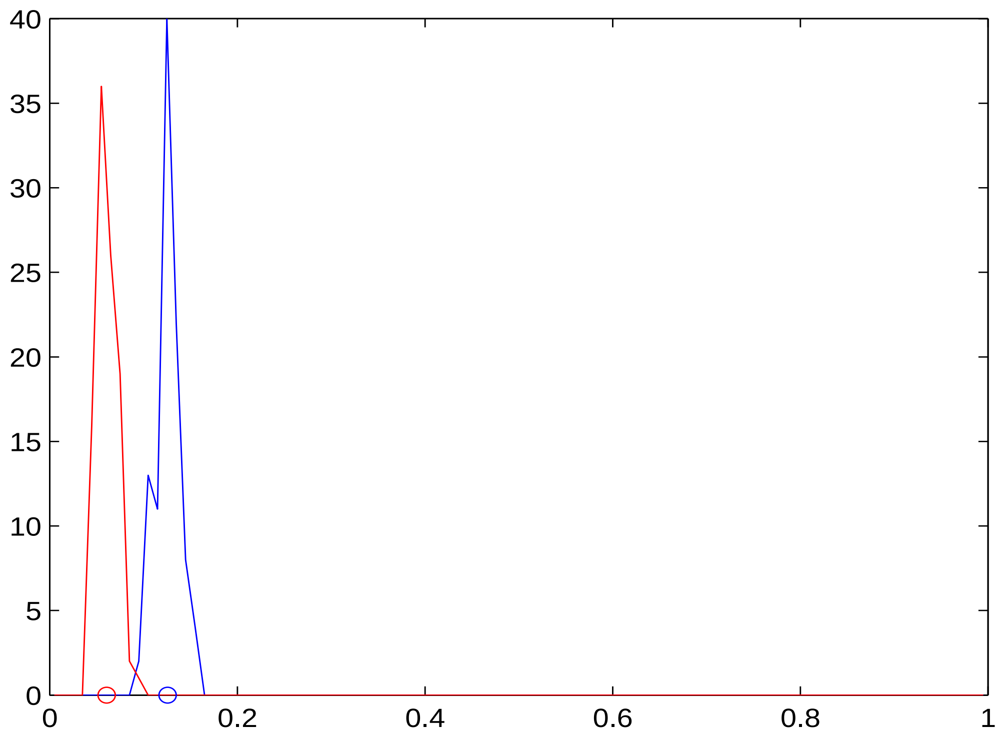
Using a kernel

- Can consider much higher dimensional spaces using the kernel trick

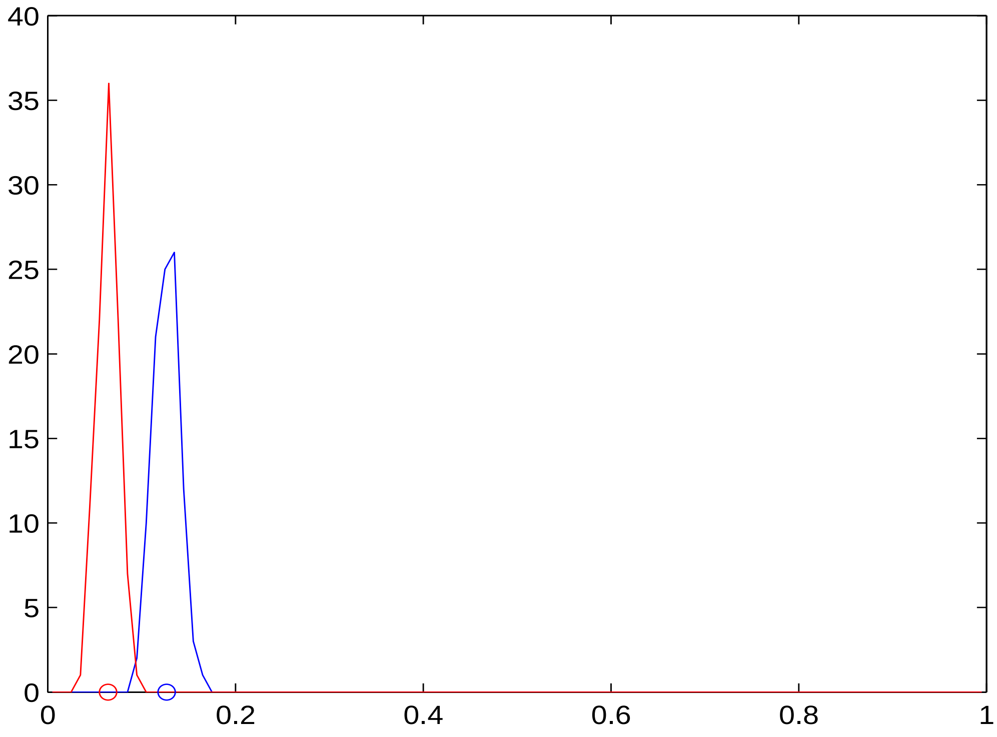
- Can even work in infinite dimensional spaces, eg using the Gaussian kernel:

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$$

Error distribution: dataset size: 342



Error distribution: dataset size: 273



Applying to 1-norm SVMs

We take the following formulation of the 1-norm SVM to optimise the bound:

$$\begin{array}{ll} \min_{\mathbf{w},b,\gamma,\xi} & -\gamma + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right) \geq \gamma - \xi_i,\ \xi_i \geq 0, \\ & i = 1,\ldots,m, \text{ and } \|\mathbf{w}\|^2 = 1. \end{array} \qquad (2)$$

Note that

$$\xi_i = \left(\gamma - y_i g(\mathbf{x}_i)\right)_+,$$

where $g(\cdot) = \langle \mathbf{w}, \phi(\cdot)\rangle + b$.

Dual form of the SVM problem

Forming the Lagrangian $L(\mathbf{w}, b, \gamma, \xi, \alpha, \beta, \lambda)$:

$$-\gamma + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \left[y_i(\langle \phi\left(\mathbf{x}_i\right), \mathbf{w} \rangle + b) - \gamma + \xi_i \right]$$
$$- \sum_{i=1}^{m} \beta_i \xi_i + \lambda \left(\|\mathbf{w}\|^2 - 1 \right)$$

with $\alpha_i \geq 0$ and $\beta_i \geq 0$.

Dual form of the SVM problem

Taking derivatives gives:

$$
\begin{aligned}
\frac{\partial L(\mathbf{w}, b, \gamma, \xi, \alpha, \beta, \lambda)}{\partial \mathbf{w}} &= 2\lambda\mathbf{w} - \sum_{i=1}^{m} y_i\alpha_i\phi\left(\mathbf{x}_i\right) = \mathbf{0}, \\
\frac{\partial L(\mathbf{w}, b, \gamma, \xi, \alpha, \beta, \lambda)}{\partial \xi_i} &= C - \alpha_i - \beta_i = 0, \\
\frac{\partial L(\mathbf{w}, b, \gamma, \xi, \alpha, \beta, \lambda)}{\partial b} &= \sum_{i=1}^{m} y_i\alpha_i = 0, \\
\frac{\partial L(\mathbf{w}, b, \gamma, \xi, \alpha, \beta, \lambda)}{\partial \gamma} &= 1 - \sum_{i=1}^{m} \alpha_i = 0.
\end{aligned}
$$

Dual form of the SVM problem

$$L(\alpha, \lambda) = -\frac{1}{4\lambda} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right) - \lambda,$$

which, again optimising with respect to λ, gives

$$\lambda^* = \frac{1}{2} \left(\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right) \right)^{1/2}$$

Dual form of the SVM problem

equivalent to maximising

$$L(\alpha) = -\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa \left(\mathbf{x}_i, \mathbf{x}_j\right),$$

subject to the constraints

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{m} \alpha_i = 1 \quad \sum_{i=1}^{m} y_i \alpha_i = 0$$

to give solution

$$\alpha_i^*, i = 1, \ldots, m$$

Dual form of the SVM problem

This is a convex quadratic programme: minimising a convex quadratic objective subject to linear constraints: convex if Hessian $\mathbf{G}$ is positive semi-definite:

$$\mathbf{G}_{ij} = y_i y_j \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

Matrix psd iff $\mathbf{u}'\mathbf{G}\mathbf{u} \geq 0$ for all $\mathbf{u}$:

$$\begin{aligned}
\mathbf{u}'\mathbf{G}\mathbf{u} &= \sum_{i,j=1}^{m} u_i u_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\
&= \left\langle \sum_{i=1}^{m} u_i y_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} u_j y_j \phi(\mathbf{x}_j) \right\rangle \\
&= \left\| \sum_{i=1}^{m} u_i y_i \phi(\mathbf{x}_i) \right\|^2 \geq 0
\end{aligned}$$

Dual form of the SVM problem

Kuhn-Tucker conditions:

$$\begin{aligned} \alpha_i \left[y_i(\langle \phi(\mathbf{x}_i), \mathbf{w} \rangle + b) - \gamma + \xi_i \right] &= 0 \\ \beta_i \xi_i &= 0 \end{aligned}$$

These imply:

- $\alpha_i \neq 0$ only if

  $$y_i(\langle \phi(\mathbf{x}_i), \mathbf{w} \rangle + b) = \gamma - \xi_i$$

  these correspond to support vectors – their margins are less than or equal to γ.

- $\xi_i \neq 0$ only if $\beta_i = 0$ implying that $\alpha_i = C$, i.e. for $0 < \alpha_i < C$ margin is exactly γ.

Dual form of the SVM problem

The solution can then be computed as:

choose $\quad i, j$ such that $-C < \alpha_i^* y_i < 0 < \alpha_j^* y_j < C$

$$
\begin{aligned}
b^* &= -0.5 \left(\sum_{k=1}^{m} \alpha_k^* y_k \kappa\left(\mathbf{x}_k, \mathbf{x}_i\right) + \sum_{k=1}^{m} \alpha_k^* y_k \kappa\left(\mathbf{x}_k, \mathbf{x}_j\right) \right) \\
f(\cdot) &= \operatorname{sgn} \left(\sum_{j=1}^{m} \alpha_j^* y_j \kappa\left(\mathbf{x}_j, \cdot\right) + b^* \right) ;
\end{aligned}
$$

Dual form of the SVM problem

We can compute the margin as follows:

$$\begin{aligned} \lambda^* &= \frac{1}{2}\left(\sum_{i,j=1}^{m} y_i y_j \alpha_i^* \alpha_j^* \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right)\right)^{1/2} \\ \gamma^* &= \left(2\lambda^*\right)^{-1}\left(\sum_{k=1}^{m} \alpha_k^* y_k \kappa\left(\mathbf{x}_k, \mathbf{x}_j\right) + b^*\right) \end{aligned}$$

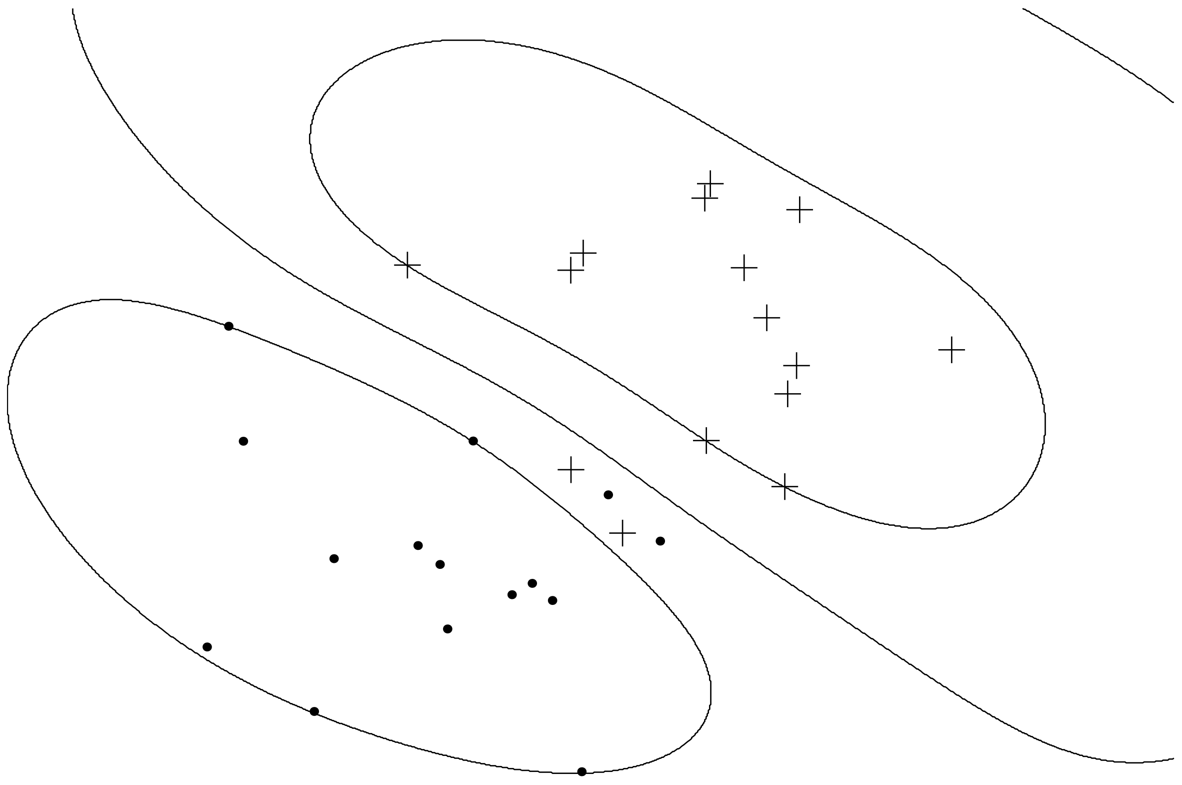
Similarly we can compute

$$\sum_{i=1}^{m} \xi_i = \frac{-2\lambda^* + \gamma^*}{C}$$

if we wish to compute the value of the bound.

Dual form of the SVM problem

Decision boundary and γ margin for 1-norm svm with a gaussian kernel:



Dual form of the SVM problem

- Have introduced a slightly non-standard version of the SVM but makes ν-SVM very simple to define.

- Consider expressing $C = 1/(\nu m)$:
  - implies $0 \leq \alpha_i \leq 1/(\nu m)$
  - if $\xi > 0$ then $\alpha_i = 1/(\nu m)$, but $\sum_{i=1}^{m} \alpha_i = 1$ so at most νm inputs can have this hold.
  - equally at least νm inputs have $\alpha_i \neq 0$

- Hence, ν can be seen as the fraction of ‘support vectors’, a natural measure of the noise in the data.

Alternative form of the SVM problem

Note more traditional form of the dual SVM optimisation:

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \kappa\left(\mathbf{x}_i, \mathbf{x}_j\right).$$

with constraints

$$0 \leq \alpha_i \leq C, \qquad \sum_{i=1}^{m} y_i \alpha_i = 0$$

Alternative form of the SVM problem

- Arises from considering renormalising so that output at margin is 1 and minimising the weight vector.
- The values of the regularisation parameter C do not correspond.
- Has advantage of simple kernel adatron algorithm if we consider the case of fixing $b = 0$ which removes the constraint $\sum_{i=1}^{m} \alpha_i y_i = 0$, so can perform gradient descent on individual α_i independently.
- SMO algorithm performs the update on pairs of α_i, α_j to ensure constraints remain satisfied.

Part 4

- Novelty detection
- Boosting
- Multiple Kernel Learning: bounds and algorithms

Novelty detection

We can also motivate novelty detection by a similar analysis as that for SVM: consider a hypersphere centred at $\mathbf{c}$ of radius r and the function g:

$$g(\mathbf{x}) = \begin{cases} 0, & \text{if } \|\mathbf{c} - \phi(\mathbf{x})\| \leq r; \\ (\|\mathbf{c} - \phi(\mathbf{x})\|^2 - r^2)/\gamma, & \text{if } r^2 \leq \|\mathbf{c} - \phi(\mathbf{x})\|^2 \leq r^2 + \gamma; \\ 1, & \text{otherwise.} \end{cases}$$

with probability at least $1 - \delta$

$$\mathbb{E}[g(\mathbf{x})] \leq \hat{\mathbb{E}}[g(\mathbf{x})] + \frac{6R^2}{\gamma\sqrt{m}} + 3\sqrt{\frac{\ln(2/\delta)}{2m}}$$

Note that tension is between creating a tight bound and defining a small sphere.

Novelty detection

Let

$$\xi_i = (\|\mathbf{c} - \phi(\mathbf{x})\|^2 - r^2)_+$$

so that

$$\hat{\mathbb{E}}[g(\mathbf{x})] \leq \frac{1}{\gamma m}\|\xi\|_1$$

Treating γ as fixed we minimise the bound by minimising $\|\xi\|_1$ and r:

$$\begin{array}{ll} \min_{\mathbf{c},r,\xi} & r^2 + C\,\|\xi\|_1 \\ \text{subject to} & \|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq r^2 + \xi_i \\ & \xi_i \geq 0,\ \ i = 1, \ldots, m \end{array}$$

Novelty detection

Again introducing the Lagrangian $L(\mathbf{c}, r, \alpha, \xi)$

$$r^2 + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i\left[\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 - \xi_i\right] - \sum_{i=1}^{m}\beta_i\xi_i.$$

Differentiating with respect to the primal variables gives:

$$\begin{aligned}
\frac{\partial L(\mathbf{c}, r, \alpha, \xi)}{\partial \mathbf{c}} &= 2\sum_{i=1}^{m}\alpha_i(\phi(\mathbf{x}_i) - \mathbf{c}) = \mathbf{0};\\
\frac{\partial L(\mathbf{c}, r, \alpha, \xi)}{\partial r} &= 2r\left(1 - \sum_{i=1}^{m}\alpha_i\right) = 0;\\
\frac{\partial L(\mathbf{c}, r, \alpha, \xi)}{\partial \xi_i} &= C - \alpha_i - \beta_i = 0.
\end{aligned}$$

Novelty detection

The final equation implies that $\alpha_i \leq C$. Substituting, we obtain

$$\begin{aligned}
L(\mathbf{c}, r, \alpha, \xi) &= r^2 + C\sum_{i=1}^{m}\xi_i \\
&\quad + \sum_{i=1}^{m}\alpha_i\left[\|\phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 - \xi_i\right] - \sum_{i=1}^{m}\beta_i\xi_i \\
&= \sum_{i=1}^{m}\alpha_i\left\langle\phi(\mathbf{x}_i) - \mathbf{c}, \phi(\mathbf{x}_i) - \mathbf{c}\right\rangle \\
&= \sum_{i=1}^{m}\alpha_i\kappa\left(\mathbf{x}_i, \mathbf{x}_i\right) - \sum_{i,j=1}^{m}\alpha_i\alpha_j\kappa\left(\mathbf{x}_i, \mathbf{x}_j\right),
\end{aligned}$$

Novelty detection

Hence optimisation maximise

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

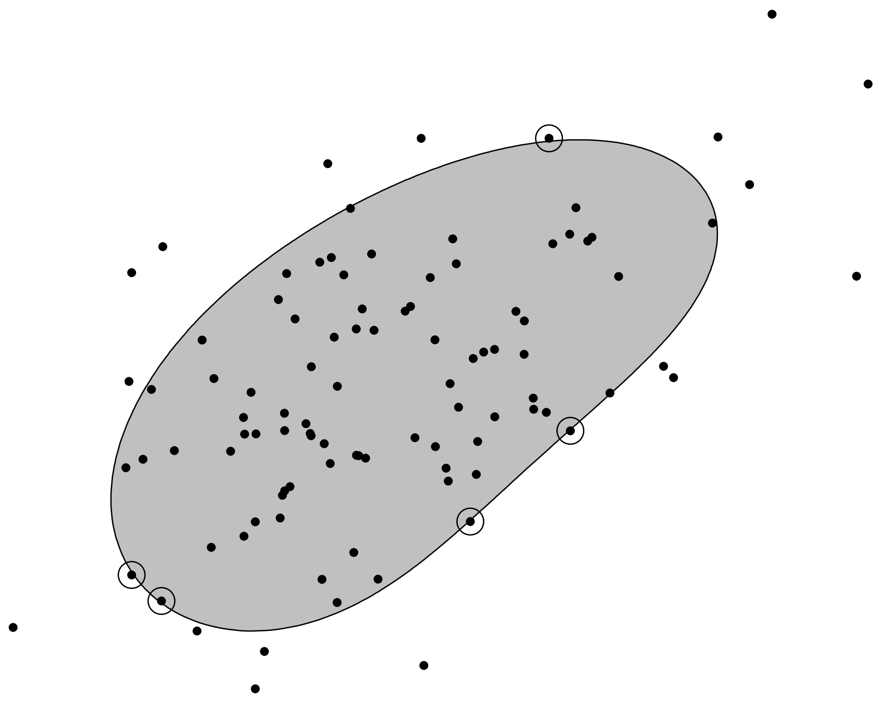
subject to $\sum_{i=1}^{m} \alpha_i = 1$ and $0 \leq \alpha_i \leq C, i = 1, \ldots, m.$
with final novelty test being:

$$f(\cdot) = \mathcal{H}\left[\kappa(\cdot,\cdot) - 2\sum_{i=1}^{m} \alpha_i^* \kappa(\mathbf{x}_i, \cdot) + D\right]$$

where

$$D = \sum_{i,j=1}^{m} \alpha_i^* \alpha_j^* \kappa(\mathbf{x}_i, \mathbf{x}_j) - (r^*)^2 - \gamma$$

Novelty detection



Final Boosting bound

- Applying a similar strategy for Boosting with the 1-norm of the slack variables we arrive at Linear programming boosting that minimises

$$\sum_h a_h + C \sum_{i=1}^m \xi_i,$$

where $\xi_i = \left(1 - y_i \sum_h a_h h(\mathbf{x}_i)\right)_+$.

- with corresponding bound:

$$\begin{aligned} P(y \neq \operatorname{sgn}(g(\mathbf{x}))) &= \mathbb{E}\left[\mathcal{H}(-yg(\mathbf{x}))\right] \\ &\leq \frac{1}{m}\sum_{i=1}^m \xi_i + \hat{R}(H)\sum_h a_h + 3\sqrt{\frac{\ln(2/\delta)}{2m}} \end{aligned}$$

Linear programming machine

- Controversy of why boosting works and relation to bagging.
- The previous bound suggests an optimisation similar to that of SVMs.
- seeks linear function in a feature space defined explicitly.
- For example using the 1-norm it seeks $\mathbf{w}$ to solve

$$\begin{array}{ll} \min_{\mathbf{w},b,\xi} & \|\mathbf{w}\|_1 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ & i = 1, \ldots, m. \end{array}$$

Linear programming boosting

- Very slight generalisation considers the features as a set $\mathbf{H}_{ij}$ of 'weak' learners (and include the constant function as one weak learner and negative of each weak learner):

$$\begin{array}{ll} \min_{\mathbf{a},\xi} & \|\mathbf{a}\|_1 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\mathbf{H}_i\mathbf{a} \geq 1-\xi_i,\ \xi_i \geq 0,\ a_i \geq 0 \\ & i = 1,\dots,m. \end{array}$$

Alternative version

- Can explicitly optimise margin with 1-norm fixed:

$$\begin{array}{ll} \max_{\rho,\mathbf{a},\xi} & \rho - D\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\mathbf{H}_i\mathbf{a} \geq \rho - \xi_i,\ \xi_i \geq 0, a_j \geq 0 \\ & \sum_{j=1}^{N} a_j = 1. \end{array}$$

- Dual has the following form:

$$\begin{array}{ll} \min_{\beta,\mathbf{u}} & \beta \\ \text{subject to} & \sum_{i=1}^{m} u_i y_i \mathbf{H}_{ij} \leq \beta,\ j = 1,\dots,N, \\ & \sum_{i=1}^{m} u_i = 1,\ 0 \leq u_i \leq D. \end{array}$$

Column generation

Can solve the dual linear programme using an iterative method:

```
1   initialise $u_i = 1/m, i = 1, \ldots, m$, $\beta = \infty$, $J = \emptyset$
2   choose $j^\star$ that maximises $f(j) = \sum_{i=1}^{m} u_i y_i \mathbf{H}_{ij}$
3   if $f(j^\star) \leq \beta$ solve primal restricted to $J$ and exit
4   $J = J \cup \{j^\star\}$
5   Solve dual restricted to set $J$ to give $u_i, \beta$
6   Go to 2
```

- Note that u_i is a distribution on the examples
- Each j added acts like an additional weak learner

- $f(j)$ is simply the weighted classification accuracy
- Hence gives ‘boosting’ algorithm - with previous weights updated satisfying error bound
- Guaranteed convergence and soft stopping criteria

Multiple kernel learning

- MKL puts a 1-norm constraint on a linear combination of kernels:

$$\left\{ \kappa(\mathbf{x}, \mathbf{z}) = \sum_{t=1}^{N} z_t \kappa_t(\mathbf{x}, \mathbf{z}) : z_t \geq 0, \sum_{t=1}^{N} z_t = 1 \right\}$$

and trains an SVM while optimizing z_t – a convex problem, c.f. group Lasso.

- obtain corresponding bound:

$$P(y \neq \operatorname{sgn}(g(\mathbf{x}))) \leq \frac{1}{m\gamma} \sum_{i=1}^{m} \xi_i + \frac{1}{\gamma} \hat{R}_m \left(\bigcup_{t=1}^{N} \mathcal{F}_t \right) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}$$

Bounding MKL

- Need a bound on

$$\hat{R}_m\left(\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t\right)$$

where $\mathcal{F}_t = \{\mathbf{x} \to \langle \mathbf{w}, \phi_{\mathbf{t}}(\mathbf{x})\rangle : \|\mathbf{w}\| \leq 1\}$.

- First note further applications of McDiarmid gives with probability $1-\delta_0$ of a random selection of σ^*:

$$\hat{R}_m(\mathcal{F}) \leq \frac{2}{m} \sup_{f\in\mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}$$

and $$\frac{2}{m} \sup_{f\in\mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) \leq \hat{R}_m(\mathcal{F}_t) + 4\sqrt{\frac{\ln(1/\delta_t)}{2m}}$$

with probability $1-\delta_t$

Bounding MKL

- Hence taking $\delta_t = \delta/2(N+1)$ for $t = 0, \ldots, N$

$$
\begin{aligned}
&\hat{R}_m\left(\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t\right) \\
&\quad\le \frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} \\
&\quad\le \frac{2}{m} \max_{1 \le t \le N} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} \\
&\quad\le \frac{2}{m} \max_{1 \le t \le N} \hat{R}_m(\mathcal{F}_t) + 8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}}
\end{aligned}
$$

with probability $1 - \delta/2$.

Bounding MKL

- This gives an overall bound on the generalisation of MKL of

$$P(y \neq \mathrm{sgn}(g(\mathbf{x}))) \quad \leq \quad \frac{1}{m\gamma}\sum_{i=1}^{m}\xi_i + \frac{2}{\gamma m}\max_{1\leq t\leq N}\mathrm{tr}(\mathbf{K}_t) + 8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}} + 3\sqrt{\frac{\ln(4/\delta)}{2m}}$$

  where $\mathbf{K}_t$ is the t-th kernel matrix.

- Bound gives only a logarithmic (additive) dependence on the number of kernels.

Linear Programming MKL

- Column generation gives efficient MKL if we can pick the best weak learner in each $\mathcal{F}_t$ efficiently:

$$\begin{aligned}
\sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} u_i y_i f(\mathbf{x}_i) &= \sup_{\mathbf{w}:\|\mathbf{w}\| \leq 1} \sum_{i=1}^{m} u_i y_i \langle \mathbf{w}, \phi_t(\mathbf{x}_i) \rangle \\
&= \sup_{\mathbf{w}:\|\mathbf{w}\| \leq 1} \left\langle \mathbf{w}, \sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i) \right\rangle \\
&= \left\| \sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i) \right\| \\
&= \sqrt{\mathbf{u}'\mathbf{Y}\mathbf{K}_t\mathbf{Y}\mathbf{u}} =: N_t
\end{aligned}$$

easily computable from the kernel matrices (note that $\mathbf{u}$ is sparse after first iteration and can also be chosen sparse at the start).

Linear Programming MKL

- The optimal weak learner from $\mathcal{F}_t$ is realised by the weight vector that achieves the supremum

$$\mathbf{w} = \frac{\sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i)}{\|\sum_{i=1}^{m} u_i y_i \phi_t(\mathbf{x}_i)\|}$$

  which has dual representation:

$$\alpha_i = \frac{1}{N_t} u_i y_i$$

- Hence, can use the linear programming boosting approach to implement multiple kernel learning.

Part 5

- Kernel design strategies.
- Kernels for text and string kernels.
- Kernels for other structures.
- Kernels from generative models.

Kernel functions

- Already seen some properties of kernels:
  - symmetric:

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle \phi(\mathbf{z}), \phi(\mathbf{x}) \rangle = \kappa(\mathbf{z}, \mathbf{x})$$

  - kernel matrices psd:

$$\begin{aligned} \mathbf{u}'\mathbf{K}\mathbf{u} &= \sum_{i,j=1}^{m} u_i u_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i), \sum_{j=1}^{m} u_j \phi(\mathbf{x}_j) \right\rangle \\ &= \left\| \sum_{i=1}^{m} u_i \phi(\mathbf{x}_i) \right\|^2 \geq 0 \end{aligned}$$

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Note that this is equivalent to all eigenvalues non-negative – recall that eigenvalues of the kernel matrix measured the sum of the squares of the projections onto the eigenvector.
- If we have uncountable domains should also have continuity, though there are exceptions to this as well.

Kernel functions

Proof outline:

- Define feature space as class of functions:

$$\mathcal{F} = \left\{ \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \cdot) \colon m \in \mathbb{N}, \mathbf{x}_i \in X, \alpha_i \in \mathbb{R}, i = 1, \ldots, m \right\}$$

- Linear space

- embedding given by

$$\mathbf{x} \longmapsto \kappa(\mathbf{x}, \cdot)$$

Kernel functions

- inner product between

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \;\; \text{and} \; g(\mathbf{x}) = \sum_{i=1}^{n} \beta_i \kappa(\mathbf{z}_i, \mathbf{x})$$

  defined as

$$\langle f, g \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \kappa(\mathbf{x}_i, \mathbf{z}_j) = \sum_{i=1}^{m} \alpha_i g(\mathbf{x}_i) = \sum_{j=1}^{n} \beta_j f(\mathbf{z}_j),$$

- well-defined

- $\langle f, f \rangle \geq 0$ by psd property.

Kernel functions

- so-called reproducing property:

$$\langle f, \phi(\mathbf{x}) \rangle = \langle f, \kappa(\mathbf{x}, \cdot) \rangle = f(\mathbf{x})$$

- implies that inner product corresponds to function evaluation – learning a function corresponds to learning a point being the weight vector corresponding to that function:

$$\langle \mathbf{w}_f, \phi(\mathbf{x}) \rangle = f(\mathbf{x})$$

Kernel constructions

For κ_1, κ_2 valid kernels, ϕ any feature map, $\mathbf{B}$ psd matrix, $a \geq 0$ and f any real valued function, the following are valid kernels:

- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z}) + \kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = a\kappa_1(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\mathbf{x}, \mathbf{z})\kappa_2(\mathbf{x}, \mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \kappa_1(\phi(\mathbf{x}),\phi(\mathbf{z}))$,
- $\kappa(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z}$.

Kernel constructions

Following are also valid kernels:

- $\kappa(\mathbf{x},\mathbf{z}) = p(\kappa_1(\mathbf{x},\mathbf{z}))$, for p any polynomial with positive coefficients.
- $\kappa(\mathbf{x},\mathbf{z}) = \exp(\kappa_1(\mathbf{x},\mathbf{z}))$,
- $\kappa(\mathbf{x},\mathbf{z}) = \exp(-\left\|\mathbf{x}-\mathbf{z}\right\|^2/(2\sigma^2))$.

Proof of third: normalise the second kernel:

$$\begin{aligned}\frac{\exp(\langle\mathbf{x},\mathbf{z}\rangle/\sigma^2)}{\sqrt{\exp(\left\|\mathbf{x}\right\|^2/\sigma^2)\exp(\left\|\mathbf{z}\right\|^2/\sigma^2)}} &= \exp\left(\frac{\langle\mathbf{x},\mathbf{z}\rangle}{\sigma^2}-\frac{\langle\mathbf{x},\mathbf{x}\rangle}{2\sigma^2}-\frac{\langle\mathbf{z},\mathbf{z}\rangle}{2\sigma^2}\right)\\ &= \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{z}\right\|^2}{2\sigma^2}\right).\end{aligned}$$

Subcomponents kernel

For the kernel $\langle \mathbf{x}, \mathbf{z} \rangle^s$ the features can be indexed by sequences

$$\mathbf{i} = (i_1, \ldots, i_n), \sum_{j=1}^{n} i_j = s$$

where

$$\phi_{\mathbf{i}}(\mathbf{x}) = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$$

A similar kernel can be defined in which all subsets of features occur:

$$\phi : \mathbf{x} \mapsto (\phi_A(\mathbf{x}))_{A \subseteq \{1,\ldots,n\}}$$

where

$$\phi_A(\mathbf{x}) = \prod_{i \in A} x_i$$

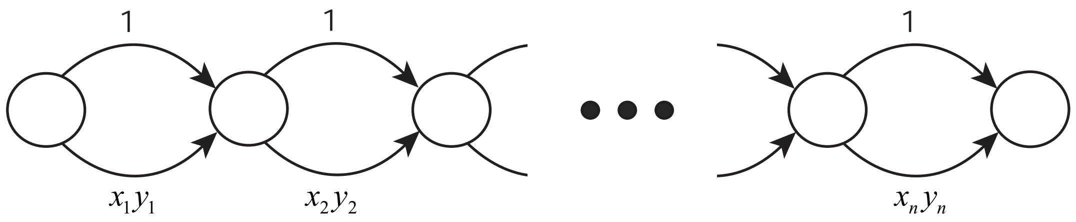
Subcomponents kernel

So we have

$$\begin{aligned}\kappa_{\subseteq}(\mathbf{x},\mathbf{y}) &= \langle \phi(\mathbf{x}), \phi(\mathbf{y})\rangle \\ &= \sum_{A\subseteq\{1,\ldots,n\}} \phi_A(\mathbf{x})\phi_A(\mathbf{y}) \\ &= \sum_{A\subseteq\{1,\ldots,n\}} \prod_{i\in A} x_i y_i = \prod_{i=1}^{n}(1+x_i y_i)\end{aligned}$$

Can represent computation with a graph:

Each path in the graph corresponds to a feature.

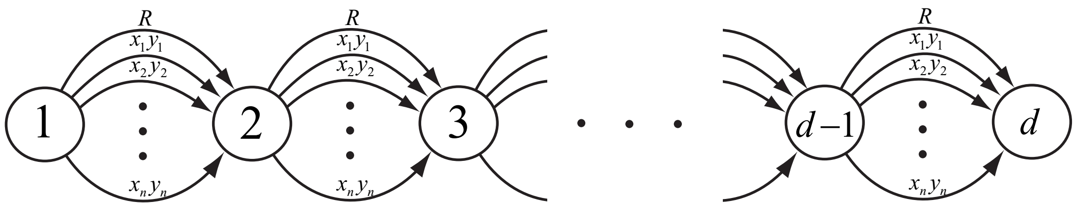


Graph kernels

Can also represent polynomial kernel

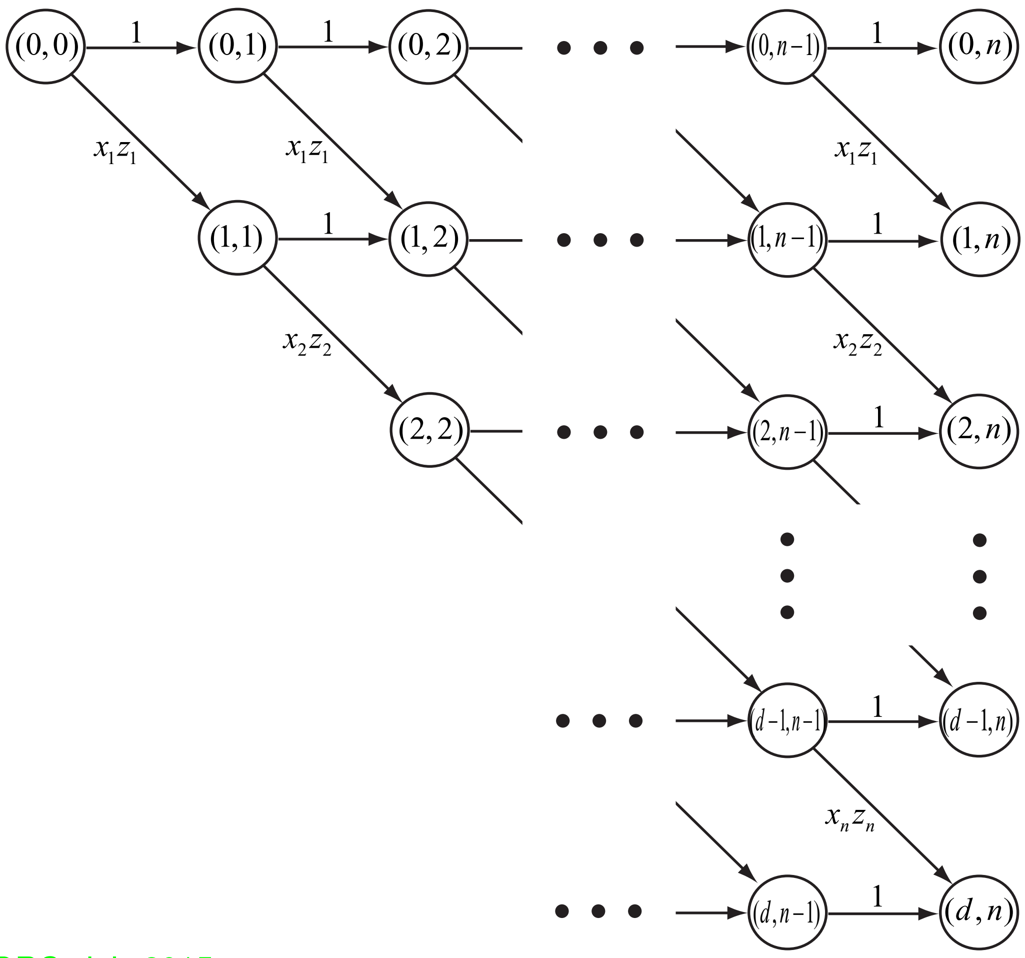
$$\kappa(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + R)^d = (x_1 y_1 + x_2 y_2 + \cdots + x_n y_n + R)^d$$

with a graph:



Graph kernels

The ANOVA kernel is represented by the graph:



Graph kernels

Features are all the combinations of exactly d distinct features, while computation is given by recursion:

$$\begin{aligned}
\kappa_0^m(\mathbf{x},\mathbf{z}) &= 1, \text{ if } m \geq 0, \\
\kappa_s^m(\mathbf{x},\mathbf{z}) &= 0, \text{ if } m < s, \\
\kappa_s^m(\mathbf{x},\mathbf{z}) &= (x_m z_m)\kappa_{s-1}^{m-1}(\mathbf{x},\mathbf{z}) + \kappa_s^{m-1}(\mathbf{x},\mathbf{z})
\end{aligned}$$

While the resulting kernel is given by

$$\kappa_d^n(\mathbf{x},\mathbf{z})$$

in the bottom right corner of the graph.

Graph kernels

- Initialise $\mathrm{DP}\left(1\right) = 1$;
- for each node compute

$$\mathrm{DP}\left(i\right) = \sum_{j \to i} \kappa_{\left(u_j \to u_i\right)}\left(\mathbf{x}, \mathbf{z}\right) \mathrm{DP}\left(j\right)$$

- result given at output node s: $\kappa(\mathbf{x}, \mathbf{z}) = \mathrm{DP}(s)$.

Kernels for text

- The simplest representation for text is the kernel given by the feature map known as the vector space model

  $$\phi : d \mapsto \phi(d) = (\mathrm{tf}(t_1, d), \mathrm{tf}(t_2, d), \ldots, \mathrm{tf}(t_N, d))'$$

  where $t_1, t_2, \ldots, t_N$ are the terms occurring in the corpus and $\mathrm{tf}(t, d)$ measures the frequency of term t in document d.

- Usually use the notation $\mathbf{D}$ for the document term matrix (cf. $\mathbf{X}$ from previous notation).

Kernels for text

- Kernel matrix is given by

$$\mathbf{K} = \mathbf{D}\mathbf{D}'$$

  wrt kernel

$$\kappa(d_1, d_2) = \sum_{j=1}^{N} \mathrm{tf}(t_j, d_1)\mathrm{tf}(t_j, d_2)$$

- despite high-dimensionality kernel function can be computed efficiently by using a linked list representation.

Semantics for text

- The standard representation does not take into account the importance or relationship between words.

- Main methods do this by introducing a ‘semantic’ mapping $\mathbf{S}$:

$$\hat{\kappa}(d_1, d_2) = \phi(d_1)'\mathbf{S}\mathbf{S}'\phi(d_2)$$

Semantics for text

- Simplest is diagonal matrix giving term weightings (known as inverse document frequency – tfidf):

$$w(t) = \ln \frac{m}{\mathrm{df}(t)}$$

- Hence kernel becomes:

$$\kappa(d_1, d_2) = \sum_{j=1}^{N} w(t_j)^2 \mathrm{tf}(t_j, d_1)\mathrm{tf}(t_j, d_2)$$

Semantics for text

- In general would also like to include semantic links between terms with off-diagonal elements, eg stemming, query expansion, wordnet.

- More generally can use co-occurrence of words in documents:
$$\mathbf{S} = \mathbf{D}'$$
so
$$(\mathbf{S}\mathbf{S}')_{ij} = \sum_{d} \mathrm{tf}(i,d)\mathrm{tf}(j,d)$$

Semantics for text

- Information retrieval technique known as latent semantic indexing uses SVD decomposition:

$$\mathbf{D}' = \mathbf{U}\mathbf{\Sigma}\mathbf{V}'$$

so that

$$d \mapsto \mathbf{U}_k'\phi(d)$$

which is equivalent to peforming kernel PCA to give latent semantic kernels:

$$\tilde{\kappa}(d_1, d_2) = \phi(d_1)'\mathbf{U}_k\mathbf{U}_k'\phi(d_2)$$

String kernels

- Consider the feature map given by

$$\phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|$$

for $u \in \Sigma^p$ with associated kernel

$$\kappa_p(s,t) = \sum_{u \in \Sigma^p} \phi_u^p(s) \phi_u^p(t)$$

String kernels

- Consider the following two sequences:

$$s = \texttt{"statistics"}$$
$$t = \texttt{"computation"}$$

The two strings contain the following substrings of length 3:

```
"sta", "tat", "ati", "tis",
"ist", "sti", "tic", "ics"
"com", "omp", "mpu", "put",
"uta", "tat", "ati", "tio", "ion"
```

and they have in common the substrings `"tat"` and `"ati"`, so their inner product would be $\kappa(s, t) = 2$.

Trie based p-spectrum kernels

- Computation organised into a trie with nodes indexed by substrings – root node by empty string ϵ.

- Create lists of substrings at root node:

  $$L_s(\epsilon) = \{(s(i : i + p - 1), 0) : i = 1, |s| - p + 1\}$$

  Similarly for t.

- Recursively through the tree: if $L_s(v)$ and $L_t(v)$ both not empty:
  for each $(u, i) \in L_*(v)$ add $(u, i + 1)$ to list $L_*(vu_{i+1})$

- At depth p increment global variable kern initialised to 0 by $|L_s(v)||L_t(v)|$.

Gap weighted string kernels

- Can create kernels whose features are all substrings of length p with the feature weighted according to all occurrences of the substring as a subsequence:

ϕ	ca	ct	at	ba	bt	cr	ar	br
cat	λ^2	λ^3	λ^2	0	0	0	0	0
car	λ^2	0	0	0	0	λ^3	λ^2	0
bat	0	0	λ^2	λ^2	λ^3	0	0	0
bar	0	0	0	λ^2	0	0	λ^2	λ^3

- This can be evaluated using a dynamic programming computation over arrays indexed by the two strings.

Tree kernels

- We can consider a feature mapping for trees defined by

$$\phi : T \longmapsto (\phi_S(T))_{S \in I}$$

  where I is a set of all subtrees and $\phi_S(T)$ counts the number of co-rooted subtrees isomorphic to the tree S.

- The computation can again be performed efficiently by working up from the leaves of the tree integrating the results from the children at each internal node.

- Similarly we can compute the inner product in the feature space given by all subtrees of the given tree not necessarily co-rooted.

Probabilistic model kernels

- There are two types of kernels that can be defined based on probabilistic models of the data.

- The most natural is to consider a class of models index by a model class M: we can then define the similarity as

$$\kappa(x, z) = \sum_{m \in M} P(x|m)P(z|m)P_M(m)$$

  also known as the marginalisation kernel.

- For the case of Hidden Markov Models this can be again be computed by a dynamic programming technique.

Probabilistic model kernels

- Pair HMMs generate pairs of symbols and under mild assumptions can also be shown to give rise to kernels that can be efficiently evaluated.
- Similarly hidden tree generating models of data, again using a recursion that works upwards from the leaves.

Fisher kernels

Fisher kernels are an alternative way of defining kernels based on probabilistic models.

- We assume the model is parametrised according to some parameters: consider the simple example of a 1-dim Gaussian distribution parametrised by μ and σ:

$$M = \left\{ P(x|\theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) : \theta = (\mu, \sigma) \in \mathbb{R}^2 \right\}.$$

- The Fisher score vector is the derivative of the log likelihood of an input x wrt the parameters:

$$\log \mathcal{L}_{(\mu,\sigma)}(x) = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{1}{2}\log(2\pi\sigma).$$

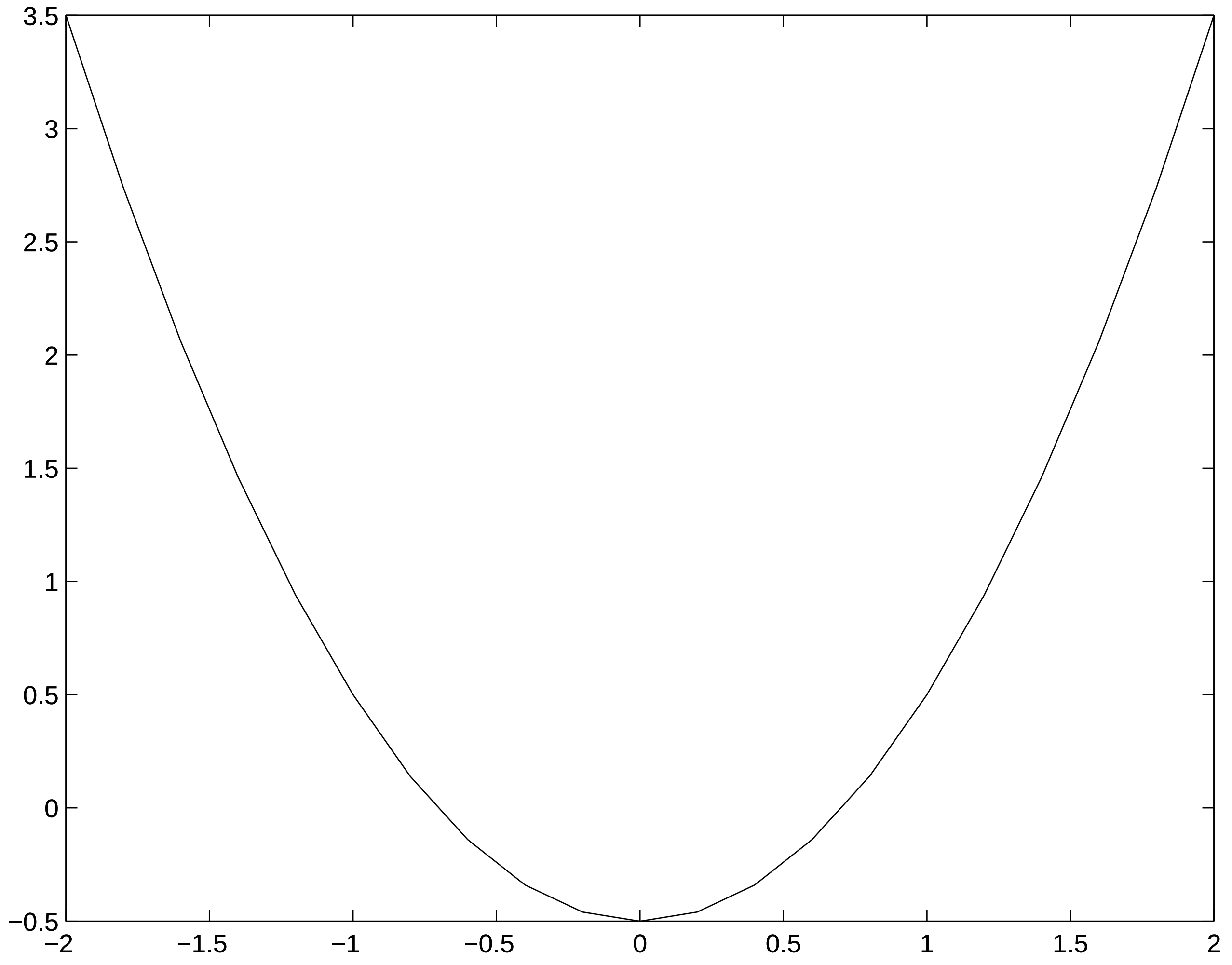
Fisher kernels

- Hence the score vector is given by:

$$\mathbf{g}\left(\theta^0, x\right) = \left(\frac{(x-\mu_0)}{\sigma_0^2}, \frac{(x-\mu_0)^2}{\sigma_0^3} - \frac{1}{2\sigma_0}\right).$$

- Taking $\mu_0 = 0$ and $\sigma_0 = 1$ the feature embedding is given by:

Fisher kernels



Fisher kernels

Can compute Fisher kernels for various models including

- ones closely related to string kernels
- Hidden Markov Models

Conclusions

Kernel methods provide a general purpose toolkit for pattern analysis

- kernels define flexible interface to the data enabling the user to encode prior knowledge into a measure of similarity between two items – with the proviso that it must satisfy the psd property.

- composition and subspace methods provide tools to enhance the representation: normalisation, centering, kernel PCA, kernel Gram-Schmidt, kernel CCA, etc.

- algorithms well-founded in statistical learning theory enable efficient and effective exploitation of the high-dimensional representations to enable good off-training performance.

Where to find out more

Web Sites: `www.support-vector.net` (SV Machines)

`www.kernel-methods.net` (kernel methods)

`www.kernel-machines.net` (kernel Machines)

`www.neurocolt.com` (Neurocolt: lots of TRs)

`www.pascal-network.org`

References

[1] N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale-sensitive Dimensions, Uniform Convergence, and Learnability. *Journal of the ACM*, 44(4):615–631, 1997.

[2] M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[3] M. Anthony and N. Biggs. *Computational Learning Theory*, volume 30 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

[4] M. Anthony and J. Shawe-Taylor. A result of Vapnik with applications. *Discrete Applied Mathematics*, 47:207–217, 1993.

[5] K. Azuma. Weighted sums of certain dependent random variables. *Tohoku Math J.*, 19:357–367, 1967.

[6] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

[7] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network.

IEEE Transactions on Information Theory, 44(2):525–536, 1998.

[8] P. L. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[9] S. Boucheron, G. Lugosi, , and P. Massart. A sharp concentration inequality with applications. *Random Structures and Algorithms*, pages vol.16, pp.277–292, 2000.

[10] O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[11] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

[12] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23–37. Springer-Verlag, 1995.

[13] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Stat. Assoc.*, 58:13–30, 1963.

[14] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[15] V. Koltchinskii and D. Panchenko. Rademacher processes and bounding the risk of function learning. *High Dimensional Probability II*, pages 443 – 459, 2000.

[16] J. Langford and J. Shawe-Taylor. PAC bayes and margins. In *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[17] M. Ledoux and M. Talagrand. *Probability in Banach Spaces: isoperimetry and processes*. Springer, 1991.

[18] C. McDiarmid. On the method of bounded differences. In 141 London Mathematical Society Lecture Notes Series, editor, *Surveys in Combinatorics 1989*, pages 148–188. Cambridge University Press, Cambridge, 1989.

[19] R. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*,

1998. (To appear. An earlier version appeared in: D.H. Fisher, Jr. (ed.), Proceedings ICML97, Morgan Kaufmann.).

[20] J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

[21] J. Shawe-Taylor and N. Cristianini. On the generalisation of soft margin algorithms. *IEEE Transactions on Information Theory*, 48(10):2721–2735, 2002.

[22] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

[23] J. Shawe-Taylor, C. Williams, N. Cristianini, and J. S. Kandola. On the eigenspectrum of the gram matrix and its relationship to the operator eigenspectrum. In *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT2002)*, volume 2533, pages 23–40, 2002.

[24] M. Talagrand. New concentration inequalities in product

spaces. *Invent. Math.*, 126:505–563, 1996.

[25] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[26] V. Vapnik and A. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Dokl. Akad. Nauk SSSR*, 181:915 – 918, 1968.

[27] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[28] Tong Zhang. Covering number bounds of certain regularized linear function classes. *Journal of Machine Learning Research*, 2:527–550, 2002.