Frontiers of Machine Learning

John Shawe-Taylor

Department of Computer Science
University College London

Data Science Summer School 2019
Pisa, September 2019

Aims:

The lectures are intended to give a personal perspective on machine learning and pattern analysis and current frontiers. We will cover:

- What is Pattern Analysis?
- Example of linear pattern functions
- Non-linearity though the kernel approach
- Approaches to deeper learning and complex applications

Aims:

The lectures are intended to give a personal perspective on machine learning and pattern analysis and current frontiers. We will cover:

- What is Pattern Analysis?
- Example of linear pattern functions
- Non-linearity though the kernel approach
- Approaches to deeper learning and complex applications

Aims:

The lectures are intended to give a personal perspective on machine learning and pattern analysis and current frontiers. We will cover:

- What is Pattern Analysis?
- Example of linear pattern functions
- Non-linearity though the kernel approach
- Approaches to deeper learning and complex applications

Aims:

The lectures are intended to give a personal perspective on machine learning and pattern analysis and current frontiers. We will cover:

- What is Pattern Analysis?
- Example of linear pattern functions
- Non-linearity though the kernel approach
- Approaches to deeper learning and complex applications

Aims:

The lectures are intended to give a personal perspective on machine learning and pattern analysis and current frontiers. We will cover:

- What is Pattern Analysis?
- Example of linear pattern functions
- Non-linearity though the kernel approach
- Approaches to deeper learning and complex applications

Pattern Analysis

- Data can exhibit regularities that may not be immediately apparent
  - exact patterns – eg motions of planets
  - complex patterns – eg genes in DNA
  - probabilistic patterns – eg market research
- Detecting patterns makes it possible to understand and/or exploit the regularities to make predictions
- Pattern analysis is the study of automatic detection of patterns in data

Pattern Analysis

- Data can exhibit regularities that may not be immediately apparent
  - exact patterns – eg motions of planets
  - complex patterns – eg genes in DNA
  - probabilistic patterns – eg market research
- Detecting patterns makes it possible to understand and/or exploit the regularities to make predictions
- Pattern analysis is the study of automatic detection of patterns in data

Pattern Analysis

- Data can exhibit regularities that may not be immediately apparent
  - exact patterns – eg motions of planets
  - complex patterns – eg genes in DNA
  - probabilistic patterns – eg market research
- Detecting patterns makes it possible to understand and/or exploit the regularities to make predictions
- Pattern analysis is the study of automatic detection of patterns in data

Defining patterns

- Exact patterns: non-trivial function f such that

$$f(\mathbf{x}) = 0 \quad \text{for all } \mathbf{x}$$

- Approximate patterns: f such that

$$f(\mathbf{x}) \approx 0 \quad \text{for all } \mathbf{x}$$

- Statistical patterns: f such that

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] \approx 0$$

where $\mathcal{D}$ is the distribution generating $\mathbf{x}$.

Defining patterns

- Exact patterns: non-trivial function f such that

$$f(\mathbf{x}) = 0 \quad \text{for all } \mathbf{x}$$

- Approximate patterns: f such that

$$f(\mathbf{x}) \approx 0 \quad \text{for all } \mathbf{x}$$

- Statistical patterns: f such that

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] \approx 0$$

where $\mathcal{D}$ is the distribution generating $\mathbf{x}$.

Defining patterns

- Exact patterns: non-trivial function f such that

$$f(\mathbf{x}) = 0 \quad \text{for all } \mathbf{x}$$

- Approximate patterns: f such that

$$f(\mathbf{x}) \approx 0 \quad \text{for all } \mathbf{x}$$

- Statistical patterns: f such that

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] \approx 0$$

where $\mathcal{D}$ is the distribution generating $\mathbf{x}$.

Pattern analysis algorithms

We would like algorithms to be:

- Computationally efficient – running time polynomial in the size of the data – often needs to be of a low degree
- Robust – able to handle noisy data, eg examples misclassified, noisy sensors or outputs only to a certain accuracy
- Statistical stability – able to distinguish between chance patterns and those characteristic of the underlying source of the data

Pattern analysis algorithms

We would like algorithms to be:

- Computationally efficient – running time polynomial in the size of the data – often needs to be of a low degree
- Robust – able to handle noisy data, eg examples misclassified, noisy sensors or outputs only to a certain accuracy
- Statistical stability – able to distinguish between chance patterns and those characteristic of the underlying source of the data

Pattern analysis algorithms

We would like algorithms to be:

- Computationally efficient – running time polynomial in the size of the data – often needs to be of a low degree
- Robust – able to handle noisy data, eg examples misclassified, noisy sensors or outputs only to a certain accuracy
- Statistical stability – able to distinguish between chance patterns and those characteristic of the underlying source of the data

Brief Historical Perspective

- Machine learning using neural like structures first considered seriously in 1960s with such systems as the Perceptron
  - Linear patterns
  - Simple learning algorithm
  - shown to be limited in complexity
- Resurrection of ideas in more powerful multi-layer perceptrons in 1980s
  - networks of perceptrons with continuous activation functions
  - very slow learning
  - no statistical analysis

Brief Historical Perspective

- Machine learning using neural like structures first considered seriously in 1960s with such systems as the Perceptron
  - Linear patterns
  - Simple learning algorithm
  - shown to be limited in complexity
- Resurrection of ideas in more powerful multi-layer perceptrons in 1980s
  - networks of perceptrons with continuous activation functions
  - very slow learning
  - no statistical analysis

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

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

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

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 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.

Kernel methods approach

- Data embedded into a Euclidean feature 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.

Kernel methods approach

- Data embedded into a Euclidean feature 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.

Kernel methods approach

- Data embedded into a Euclidean feature 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.

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

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

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:

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

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

where

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

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 between datapoints

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 between datapoints

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 between datapoints

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 we remain linear 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 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 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?

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?

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?

Theories of learning

- Basic approach of SLT is to view learning from a statistical viewpoint.
- Aim of any theory is to model real/ artificial phenomena so that we can better understand/ predict/ exploit them.
- SLT is just one approach to understanding/ predicting/ exploiting learning systems, others include Bayesian inference, inductive inference, statistical physics, traditional statistical analysis.

Theories of learning

- Basic approach of SLT is to view learning from a statistical viewpoint.
- Aim of any theory is to model real/ artificial phenomena so that we can better understand/ predict/ exploit them.
- SLT is just one approach to understanding/ predicting/ exploiting learning systems, others include Bayesian inference, inductive inference, statistical physics, traditional statistical analysis.

Theories of learning

- Basic approach of SLT is to view learning from a statistical viewpoint.
- Aim of any theory is to model real/ artificial phenomena so that we can better understand/ predict/ exploit them.
- SLT is just one approach to understanding/ predicting/ exploiting learning systems, others include Bayesian inference, inductive inference, statistical physics, traditional statistical analysis.

Theories of learning cont.

- Each theory makes assumptions about the phenomenon of learning and based on these derives predictions of behaviour as well as algorithms that aim at optimising the predictions.
- Each theory has strengths and weaknesses – the better it captures the key details of real world system, the better the theory and the better the chances of it making accurate predictions and driving good algorithms.

Theories of learning cont.

- Each theory makes assumptions about the phenomenon of learning and based on these derives predictions of behaviour as well as algorithms that aim at optimising the predictions.
- Each theory has strengths and weaknesses – the better it captures the key details of real world system, the better the theory and the better the chances of it making accurate predictions and driving good algorithms.

General statistical considerations

- Statistical models (not including Bayesian) begin with an assumption that the data is generated by an underlying distribution $\mathcal{D}$ typically not given explicitly to the learner.
- If we are trying to classify cancerous tissue from healthy tissue, there are two distributions, one for cancerous cells and one for healthy ones.

General statistical considerations

- Statistical models (not including Bayesian) begin with an assumption that the data is generated by an underlying distribution $\mathcal{D}$ typically not given explicitly to the learner.
- If we are trying to classify cancerous tissue from healthy tissue, there are two distributions, one for cancerous cells and one for healthy ones.

General statistical considerations cont.

- Usually the distribution subsumes the processes of the natural/artificial world that we are studying.
- Rather than accessing the distribution directly, statistical learning typically assumes that we are given a 'training sample' or 'training set'

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

generated identically and independently (i.i.d.) according to the distribution $\mathcal{D}$.

General statistical considerations cont.

- Usually the distribution subsumes the processes of the natural/artificial world that we are studying.
- Rather than accessing the distribution directly, statistical learning typically assumes that we are given a 'training sample' or 'training set'

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

generated identically and independently (i.i.d.) according to the distribution $\mathcal{D}$.

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:

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

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

Generalisation of a learner

- 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:

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

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.

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 by the weight vector

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

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 I

- We consider the Breast Cancer dataset from the UCI repository.
- Use the simple Parzen window classifier described by the weight vector

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

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

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

  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 all the examples, but then we plot for training set sizes:

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

Example of Generalisation II

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

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

  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 all the examples, but then we plot for training set sizes:

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

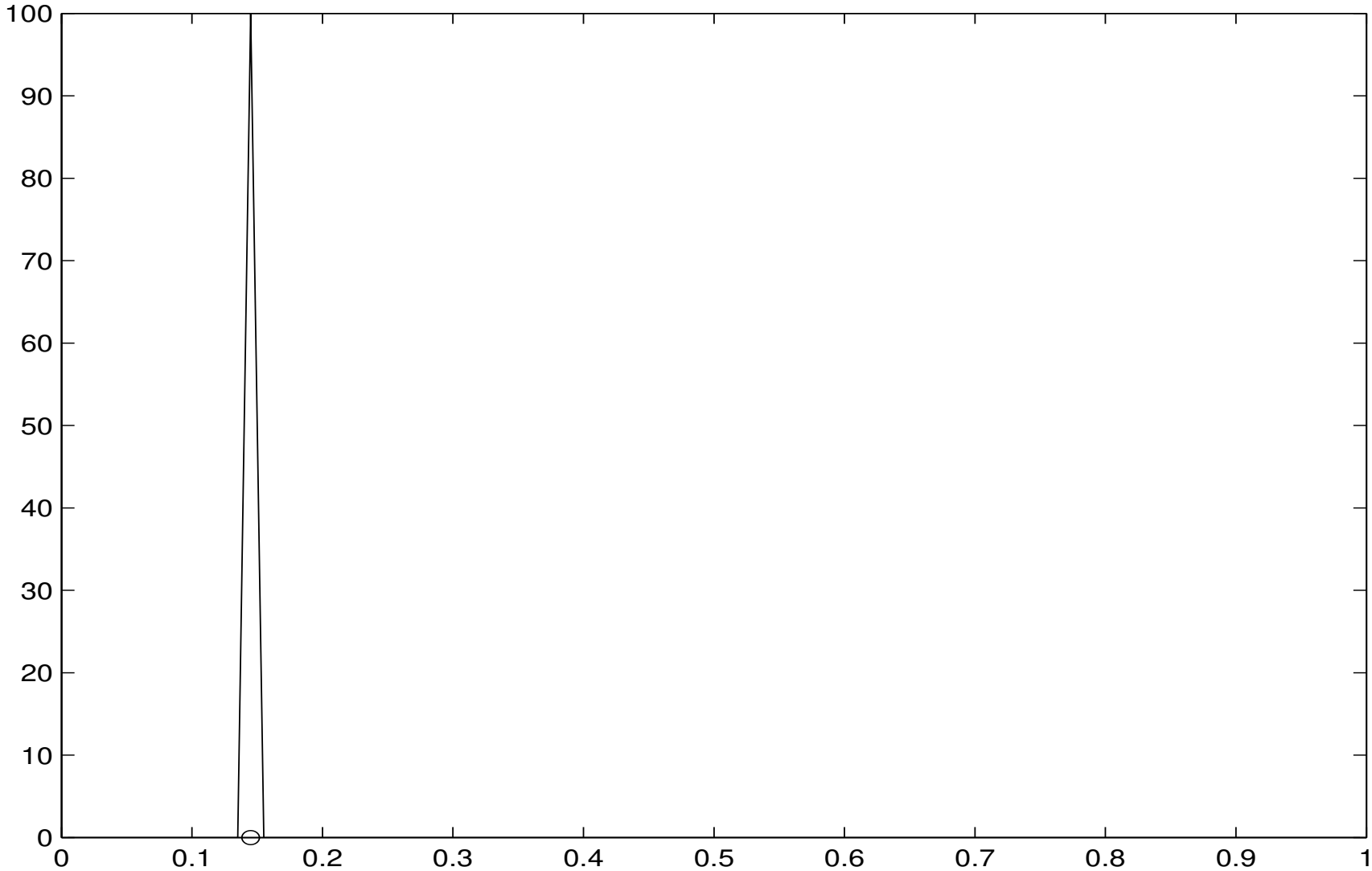
Example of Generalisation III

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

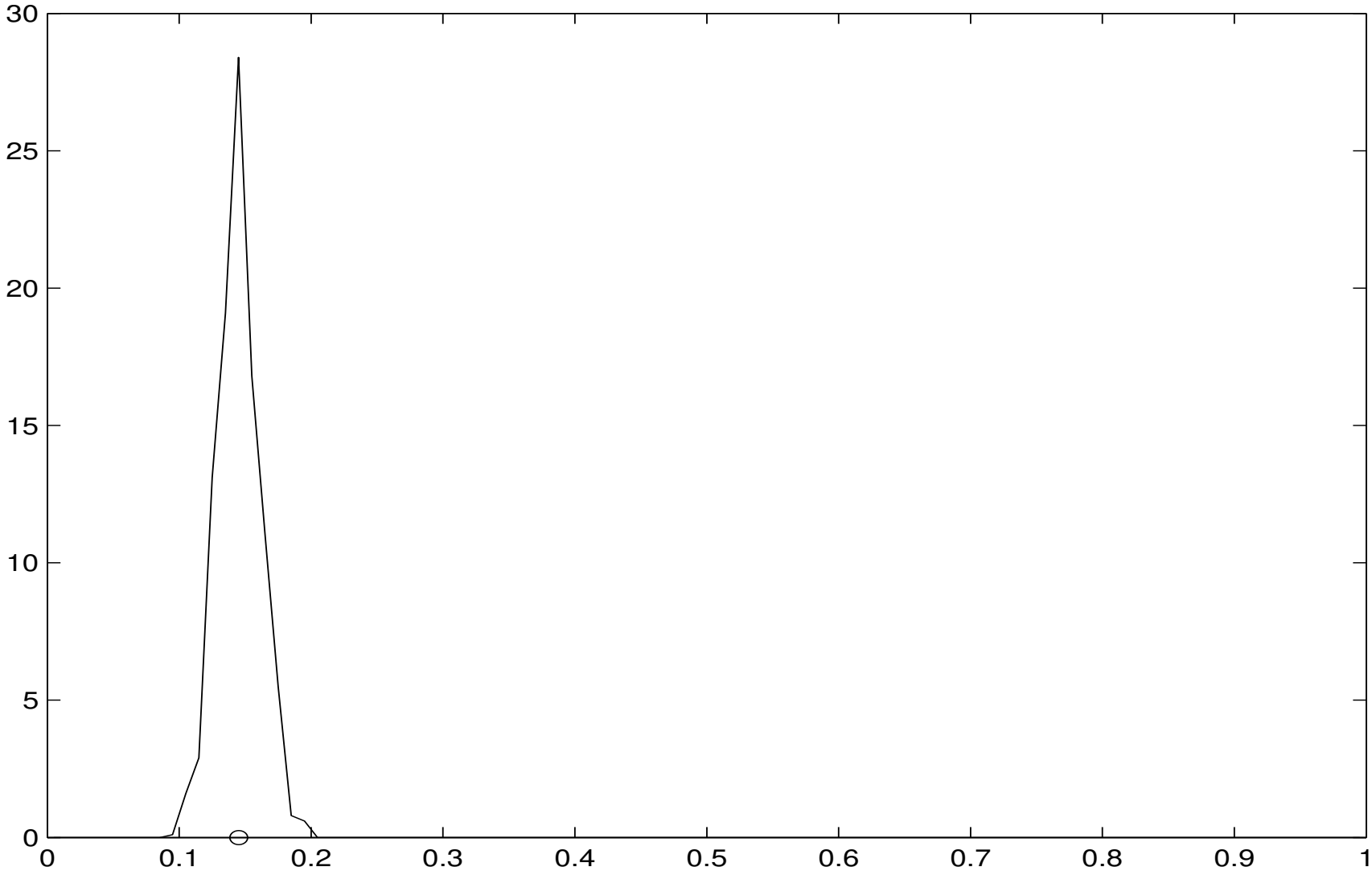
$$\begin{aligned}\mathbb{E}\left[\mathcal{A}_{\mathcal{F}}(S)\right] &= \mathbb{E}_S\left[\mathbf{w}_S^+ - \mathbf{w}_S^-\right] \\ &= \mathbb{E}_S\left[\mathbf{w}_S^+\right] - \mathbb{E}_S\left[\mathbf{w}_S^-\right] \\ &= \mathbb{E}_{y=+1}\left[\mathbf{x}\right] - \mathbb{E}_{y=-1}\left[\mathbf{x}\right],\end{aligned}$$

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.

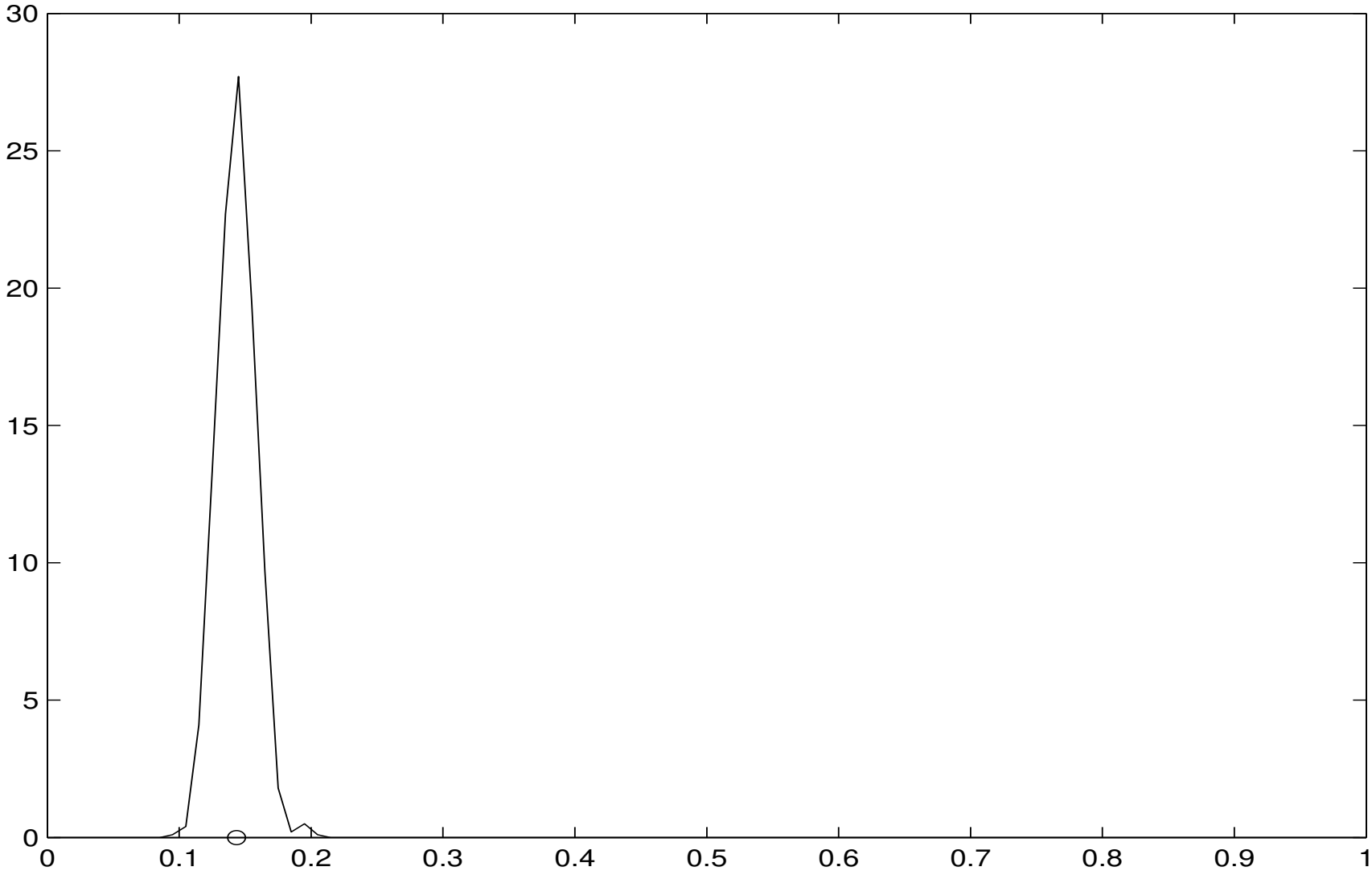
Error distribution: full dataset



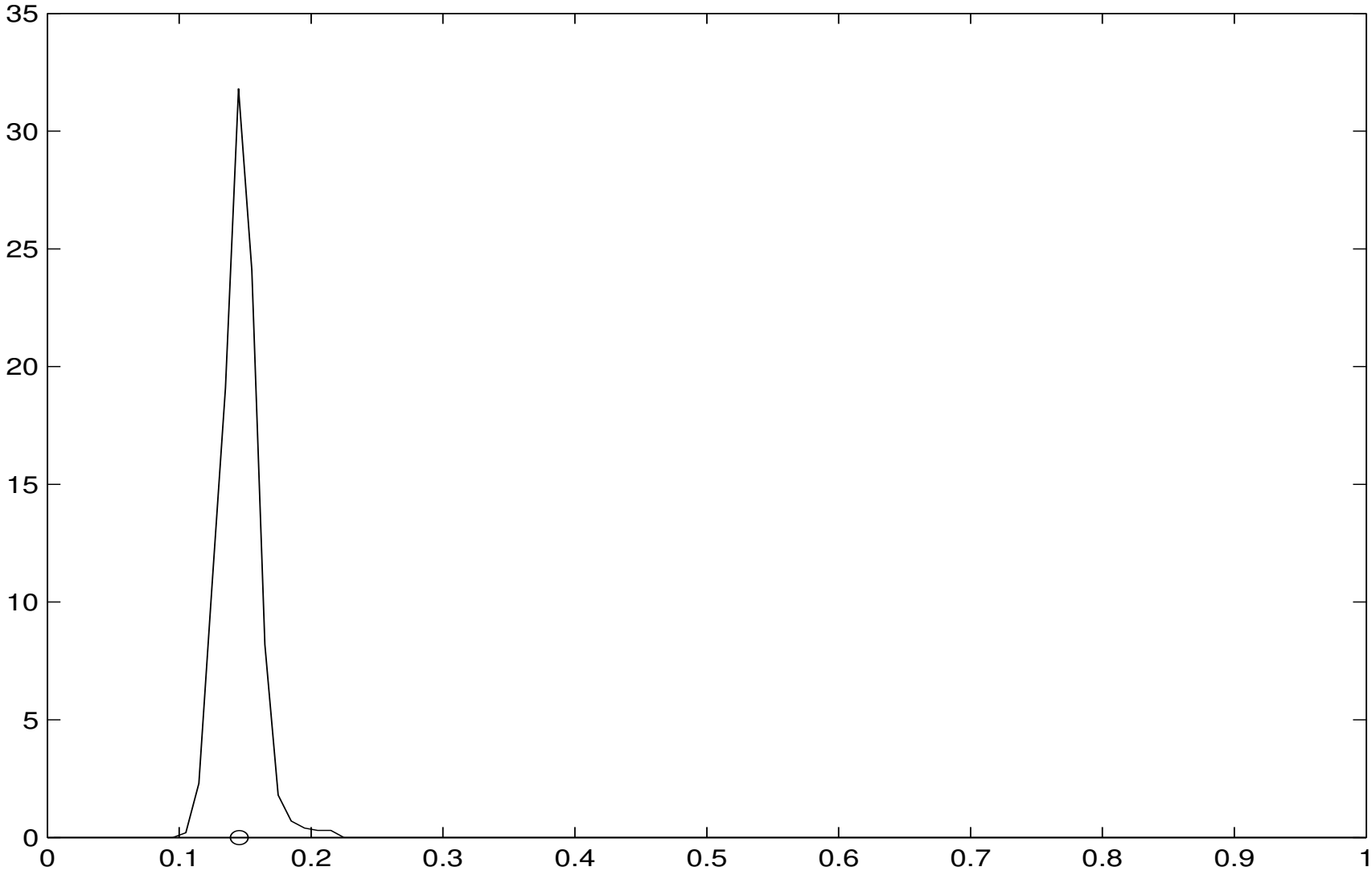
Error distribution: dataset size: 342



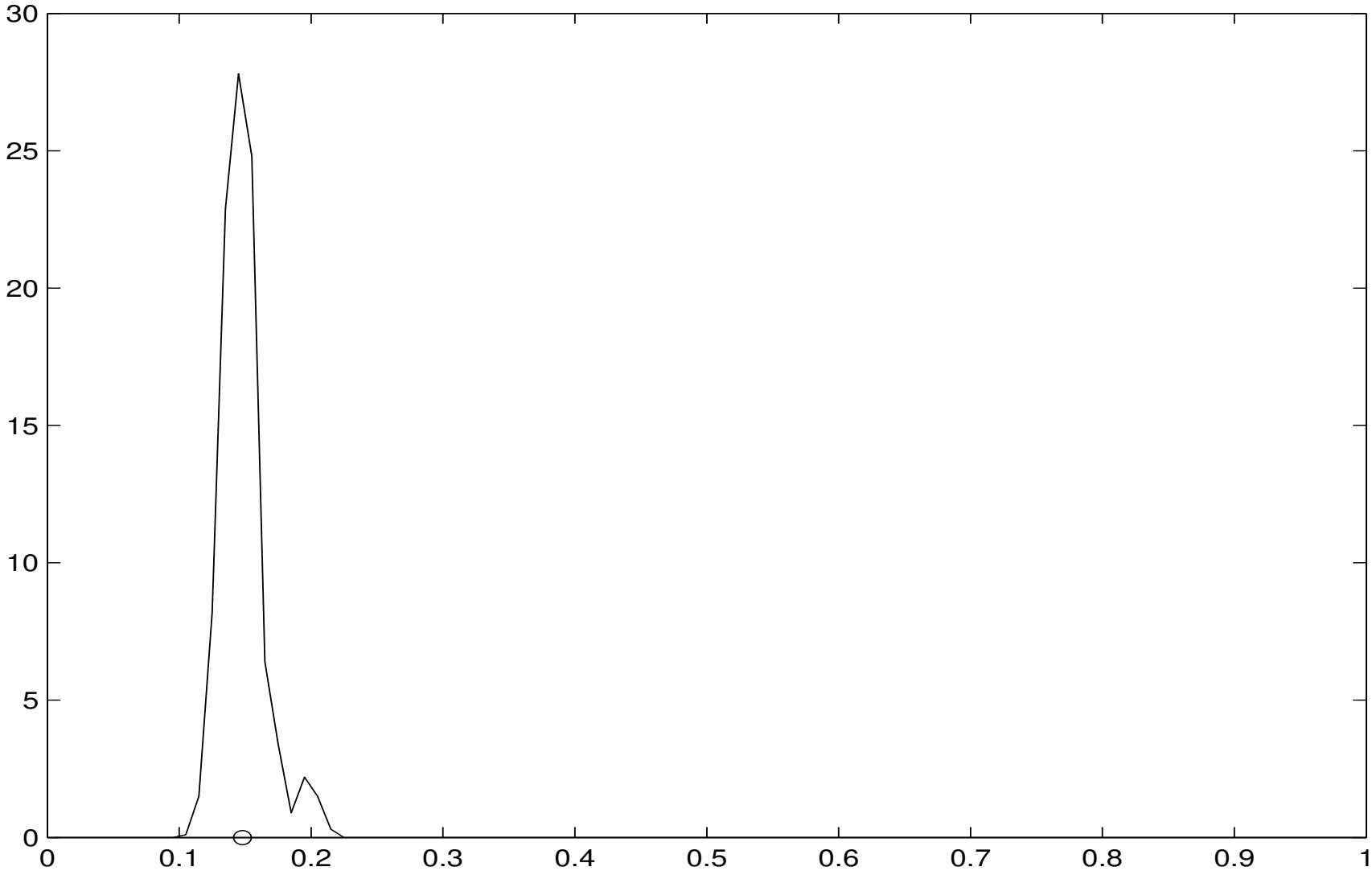
Error distribution: dataset size: 273



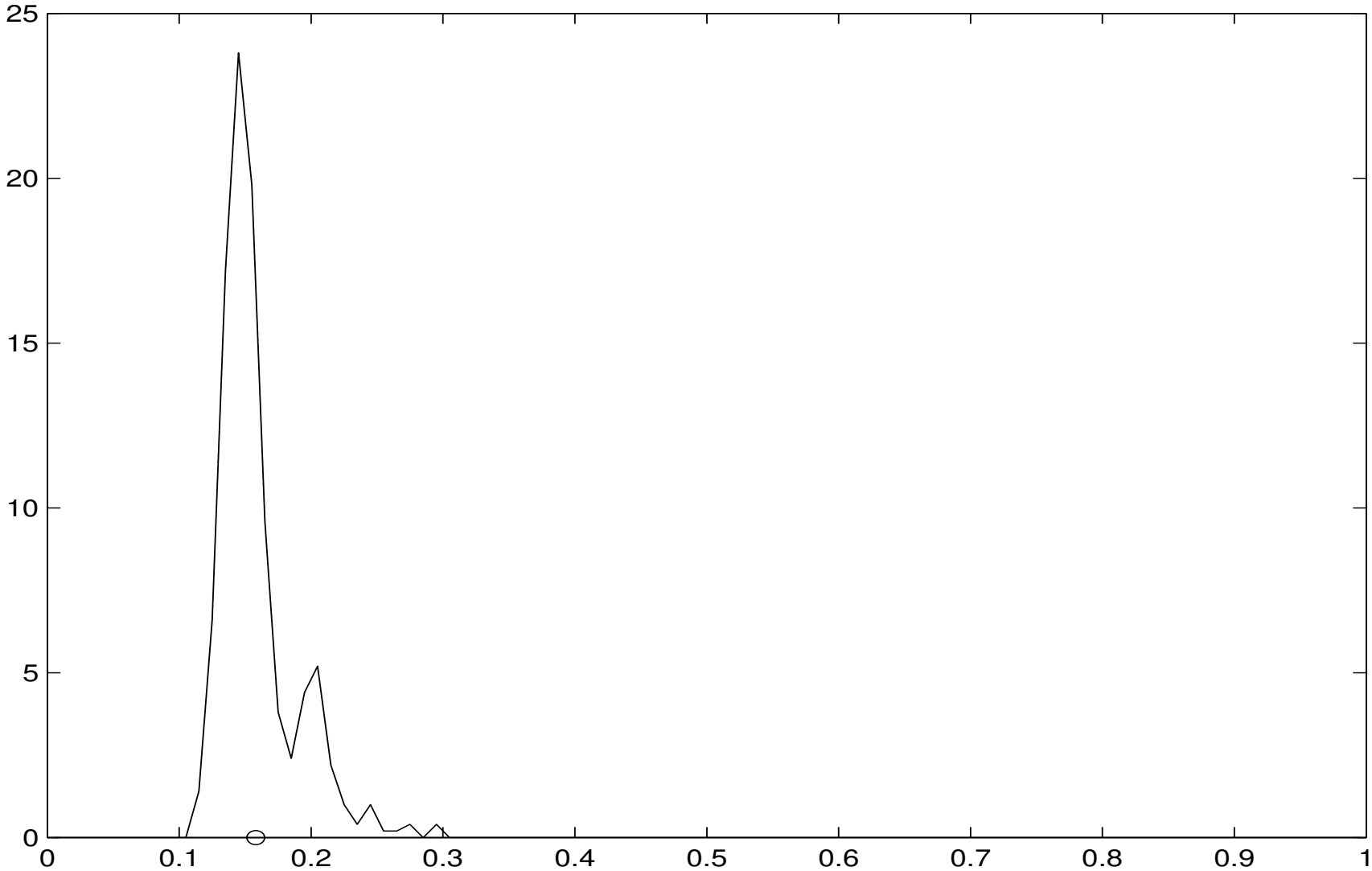
Error distribution: dataset size: 205



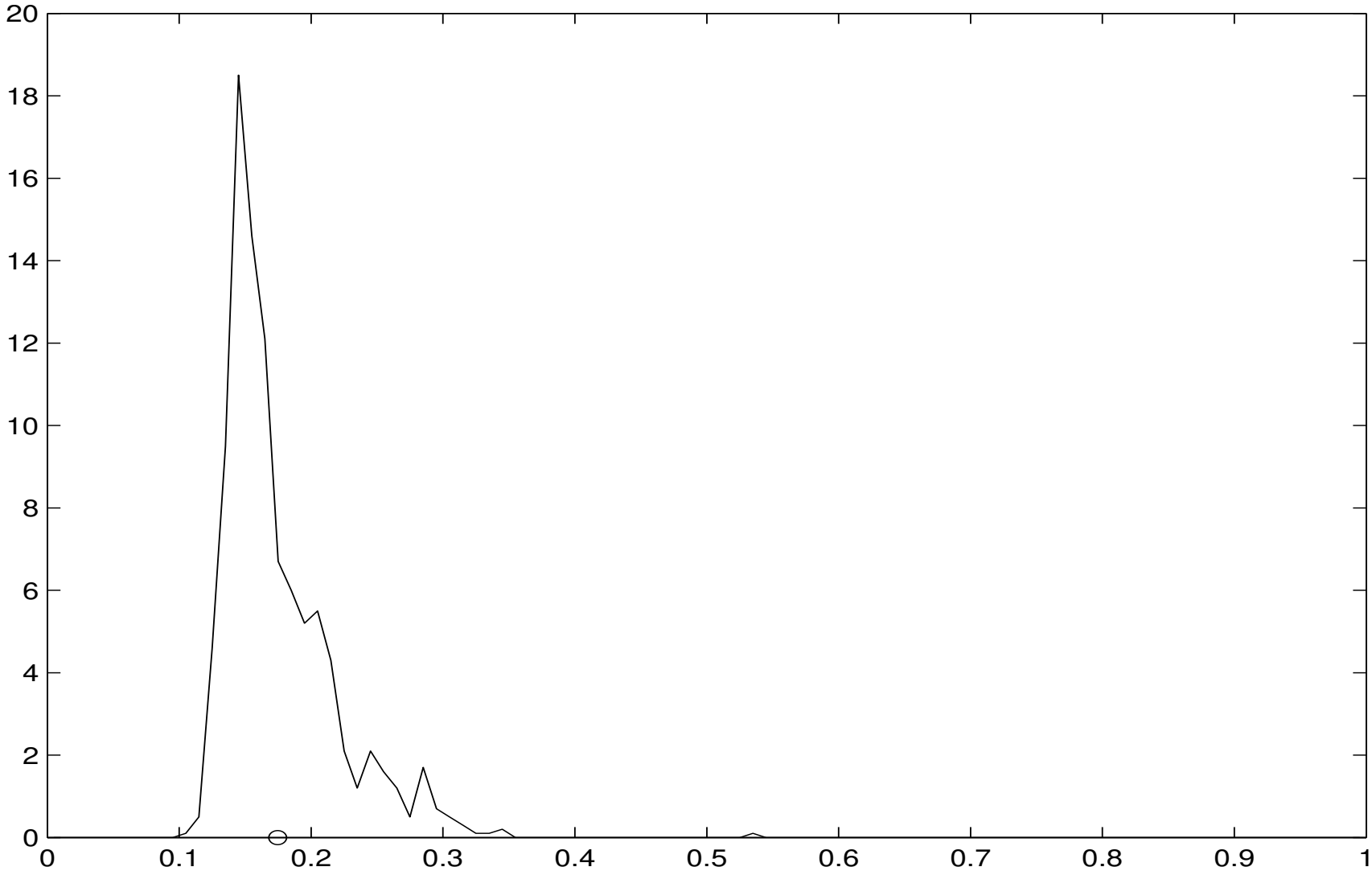
Error distribution: dataset size: 137



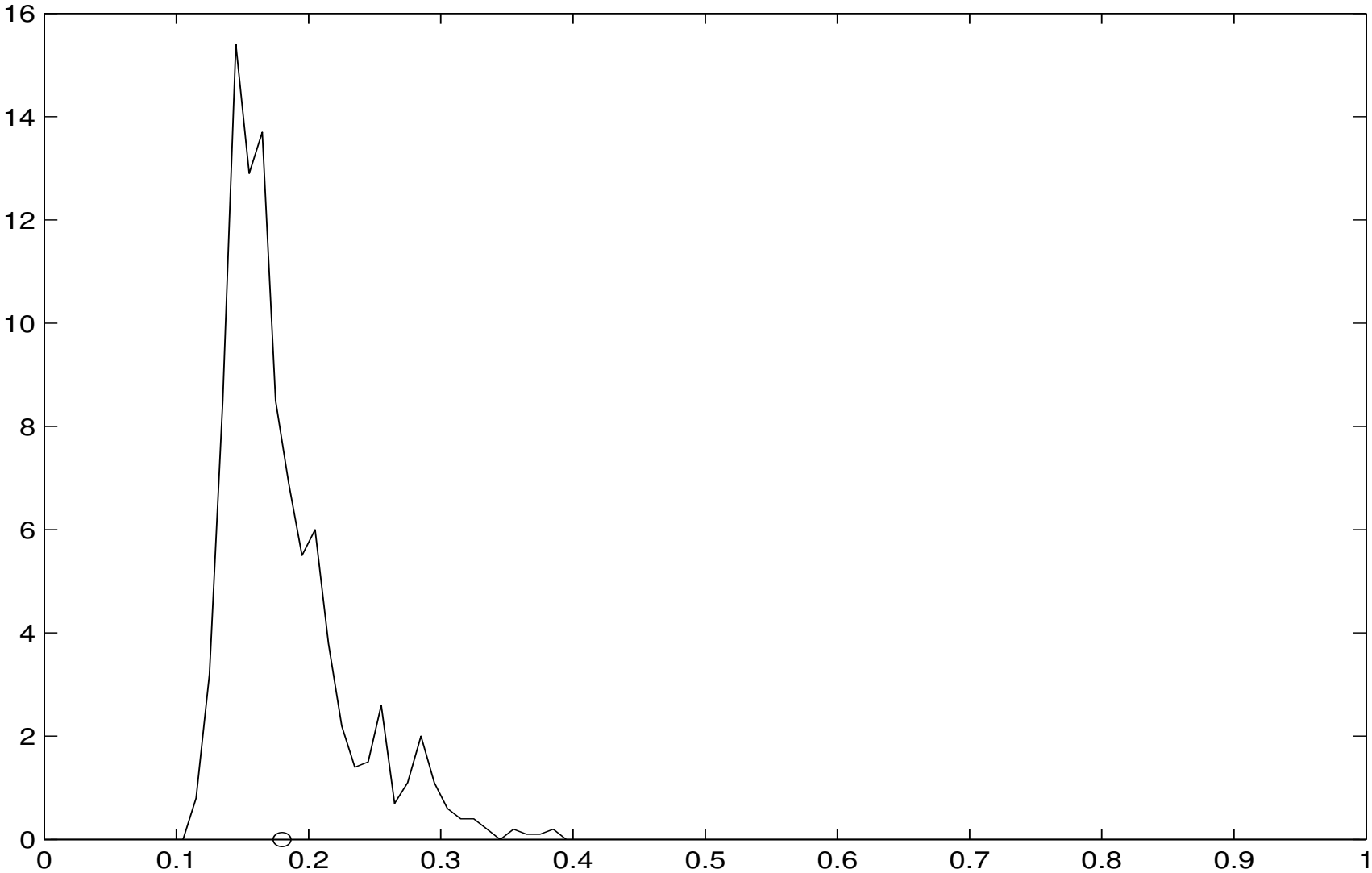
Error distribution: dataset size: 68



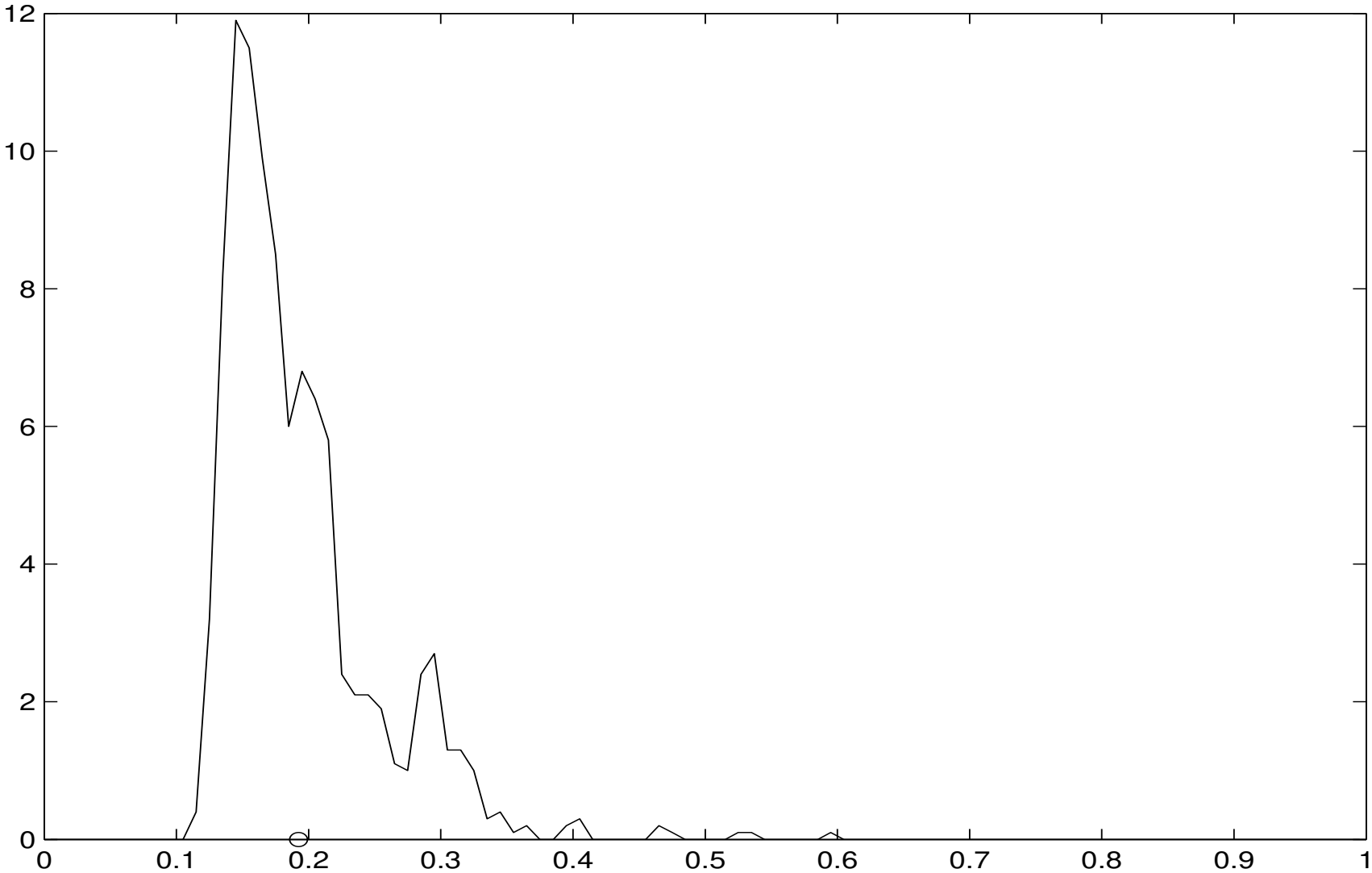
Error distribution: dataset size: 34



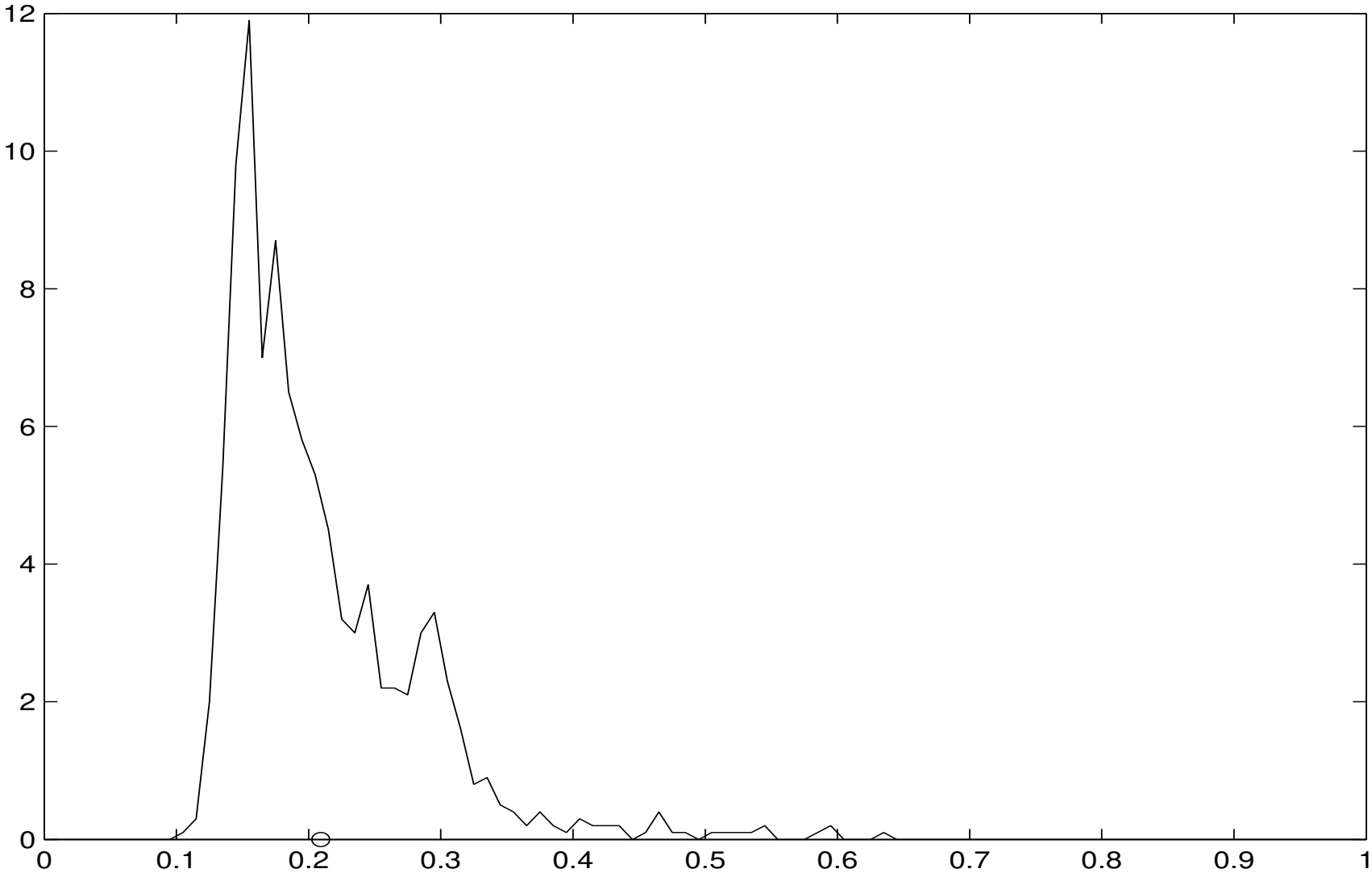
Error distribution: dataset size: 27



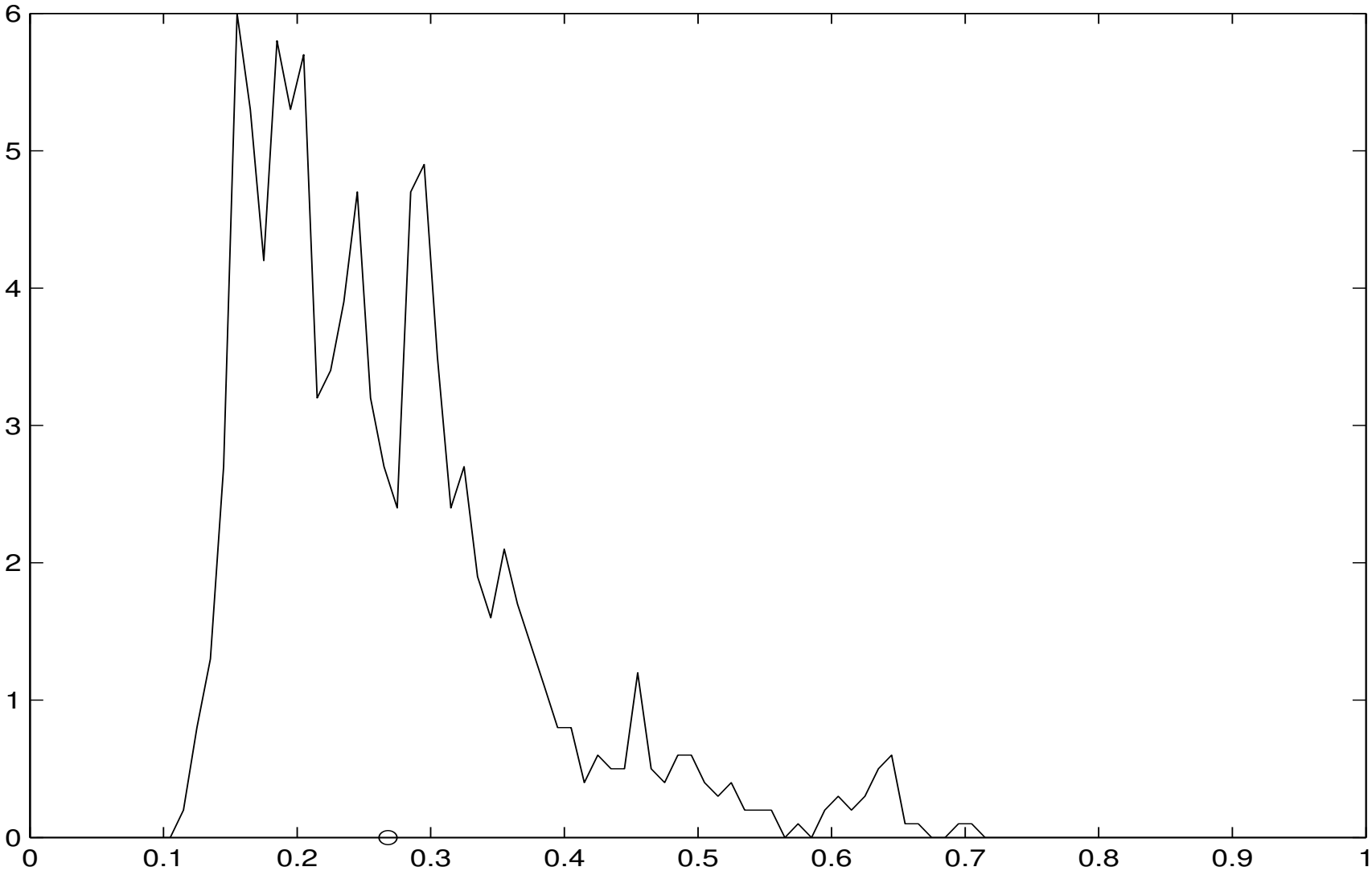
Error distribution: dataset size: 20



Error distribution: dataset size: 14



Error distribution: dataset size: 7



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

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 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, i.e. for the majority of training sets.
- 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.

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, i.e. for the majority of training sets.
- 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.

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, i.e. for the majority of training sets.
- 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

- 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 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):

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

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 $f(x_1, \ldots, x_n) = S_n$

McDiarmid's inequality

Theorem

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 $1 - \delta$

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

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

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

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 $1 - \delta$

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

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

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

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:

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

is known as the Rademacher complexity of the function class $\mathcal{F}$ where $\sigma_i, i = 1, m$ are uniformly random $+1, -1$ valued variables.

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.

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.

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

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

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

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

Application to large margin classification

- Rademacher complexity comes into its own for analysing Support Vector Machines as well as Boosting algorithms.

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

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

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

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

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

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

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

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

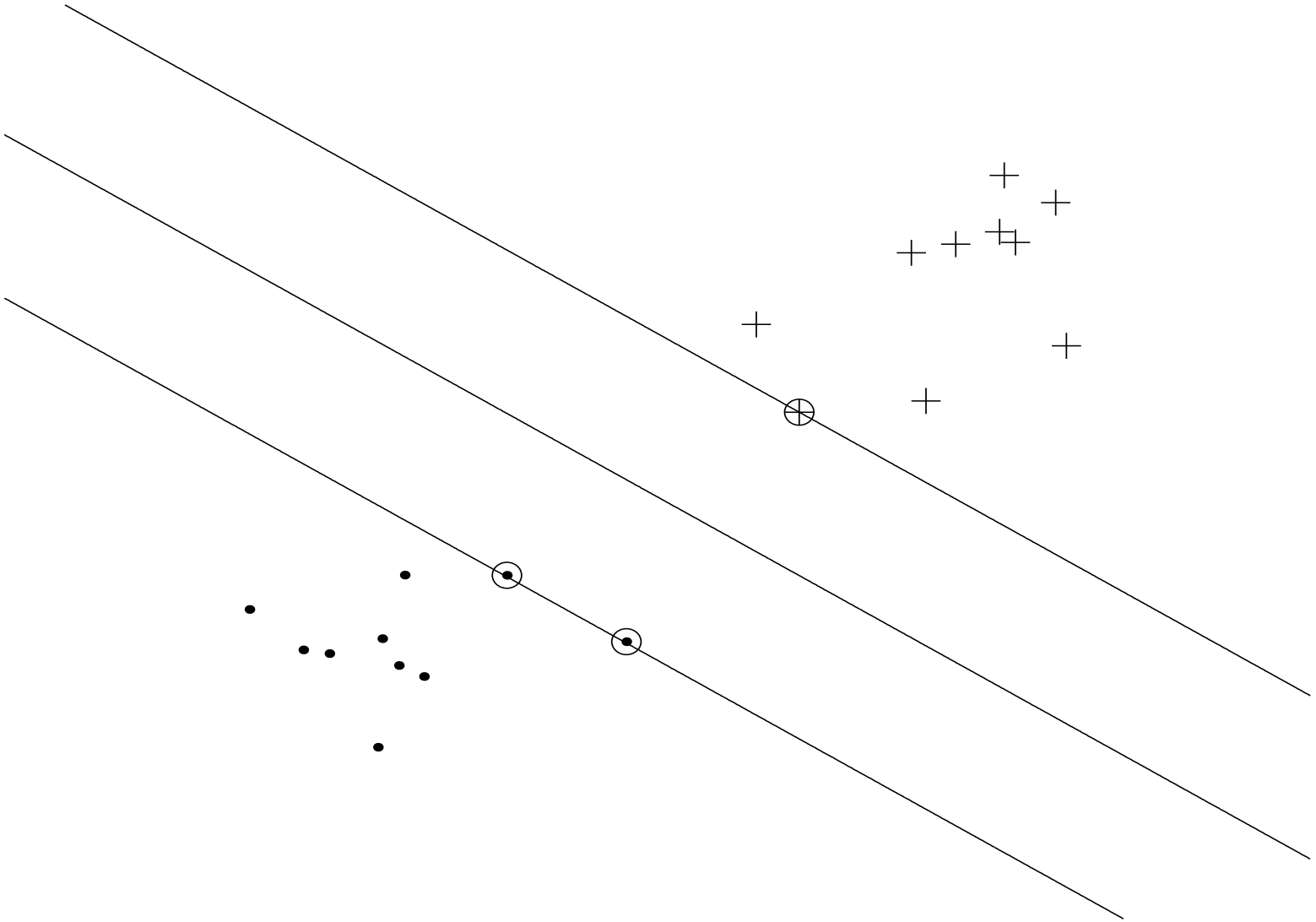
positive if correctly classified, and measures distance from the separating hyperplane when the weight vector is normalised.

- 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

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

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

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

Applying the Rademacher theorem

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

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

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

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

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 $\mathcal{A}$: $\hat{R}_m(\mathcal{A}\circ\mathcal{F}) \leq \frac{2}{\gamma}\hat{R}_m(\mathcal{F})$.

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 $\mathcal{A}$: $\hat{R}_m(\mathcal{A}\circ\mathcal{F}) \leq \frac{2}{\gamma}\hat{R}_m(\mathcal{F})$.

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:

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

- Note that for the Gaussian kernel this reduces to

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

Final SVM bound

- Assembling the result we obtain:

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

- Note that for the Gaussian kernel this reduces to

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

Applying to 1-norm SVMs

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

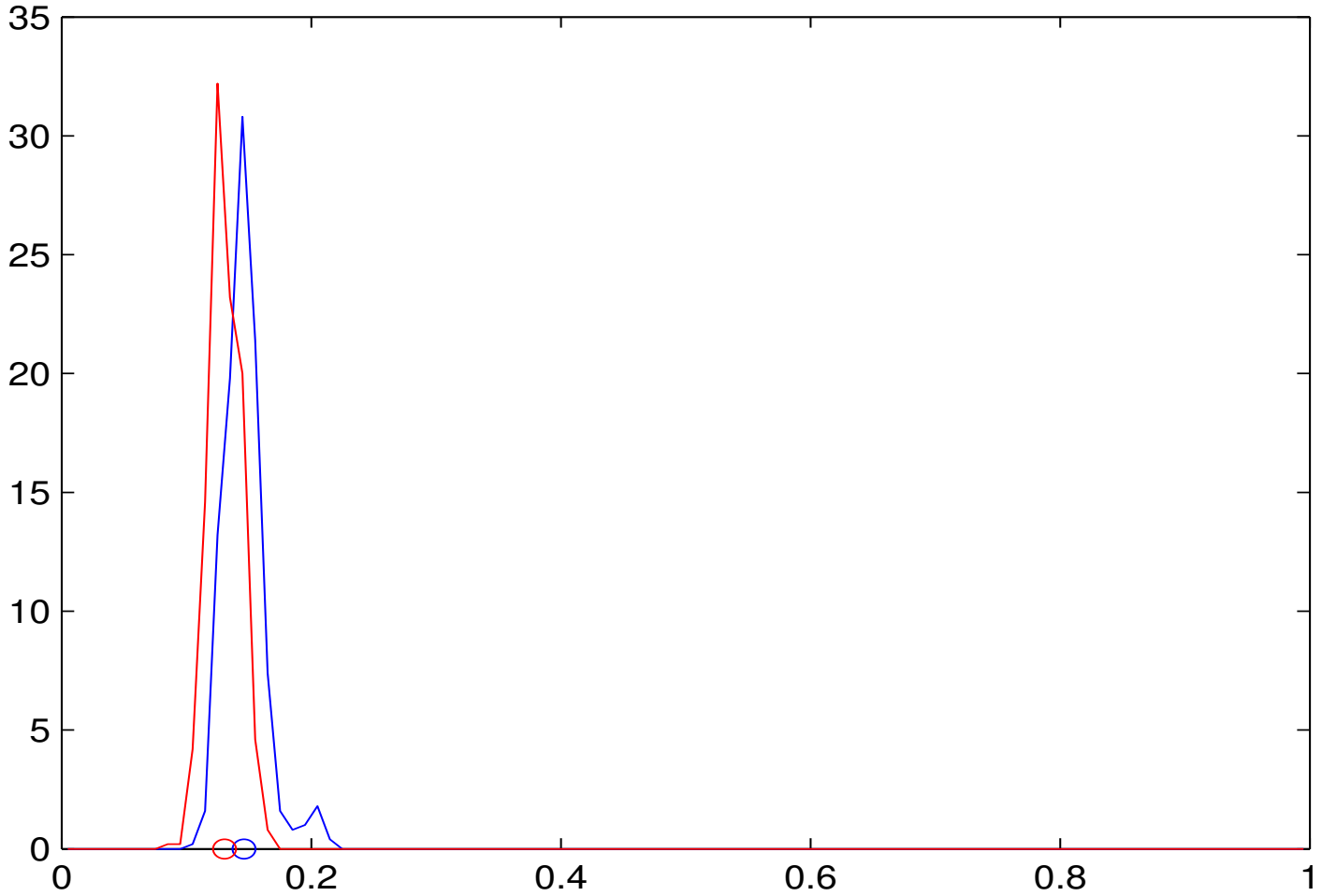
$$\begin{array}{ll} \min_{\mathbf{w},b,\gamma,\xi} & -\gamma + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} & y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right) \geq \gamma - \xi_i,\ \xi_i \geq 0, \\ & i = 1, \ldots, m, \text{ and } \|\mathbf{w}\|^2 = 1. \end{array} \tag{2}$$

Note that

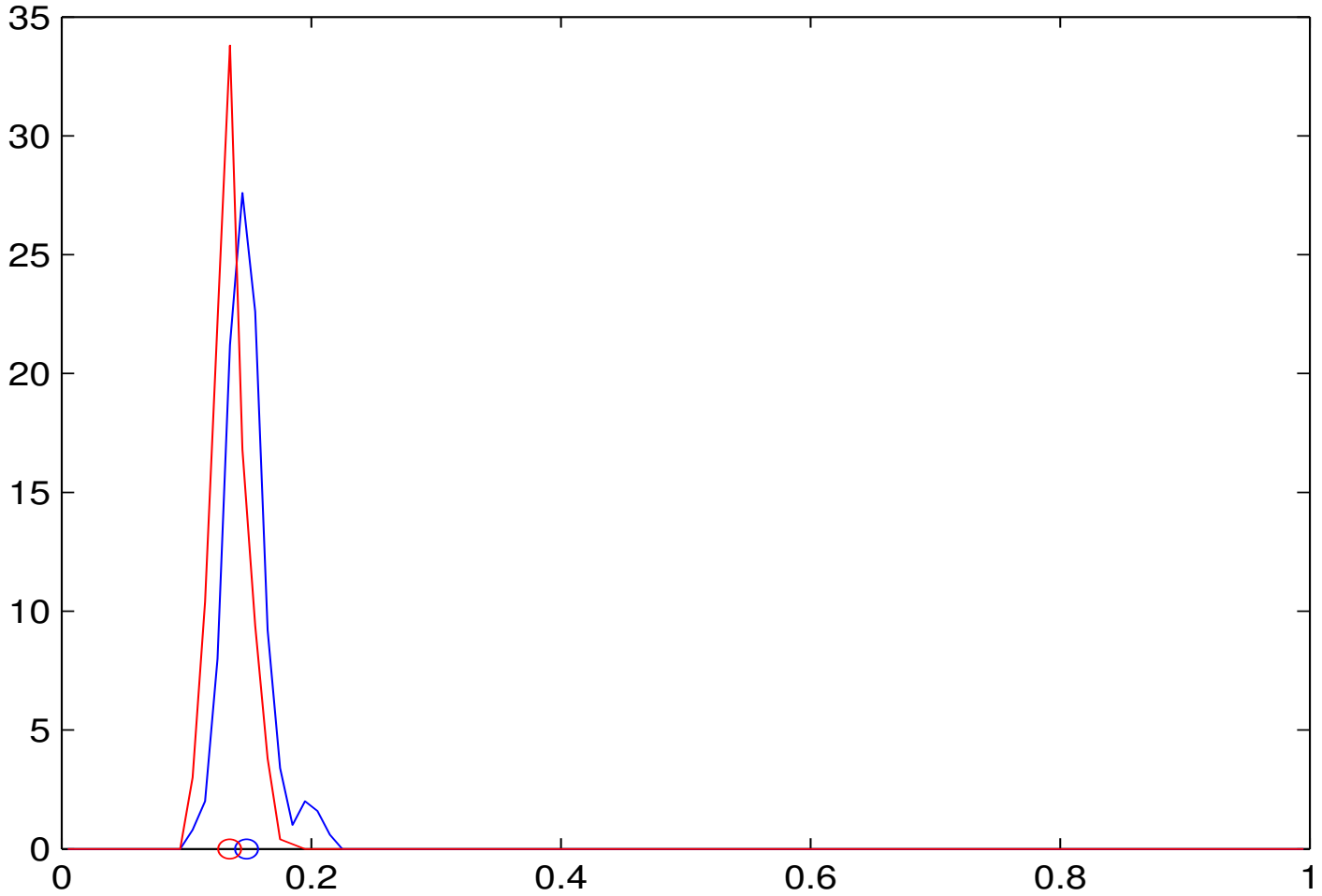
$$\xi_i = \left(\gamma - y_i g(\mathbf{x}_i)\right)_+,$$

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

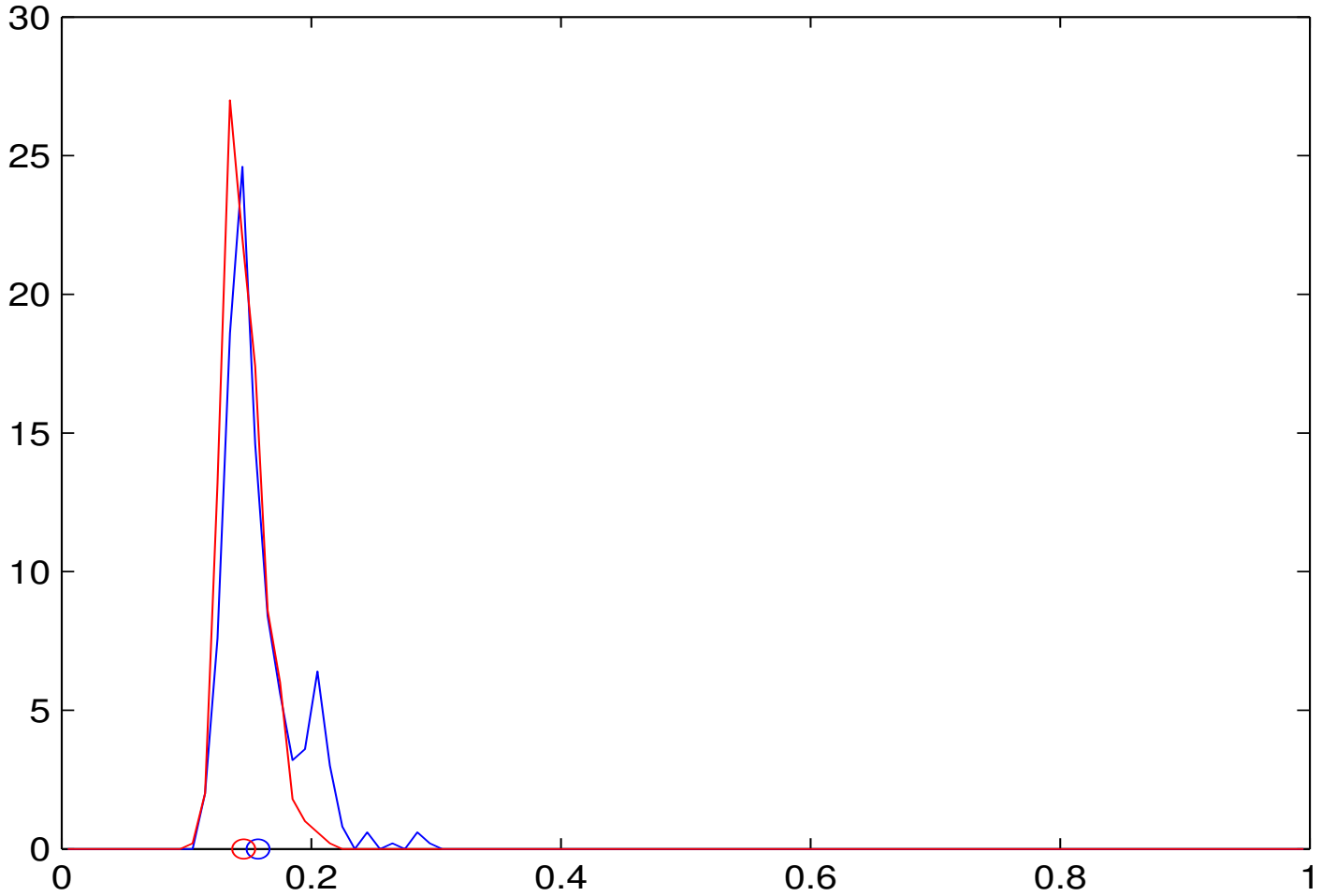
Error distribution: dataset size: 205



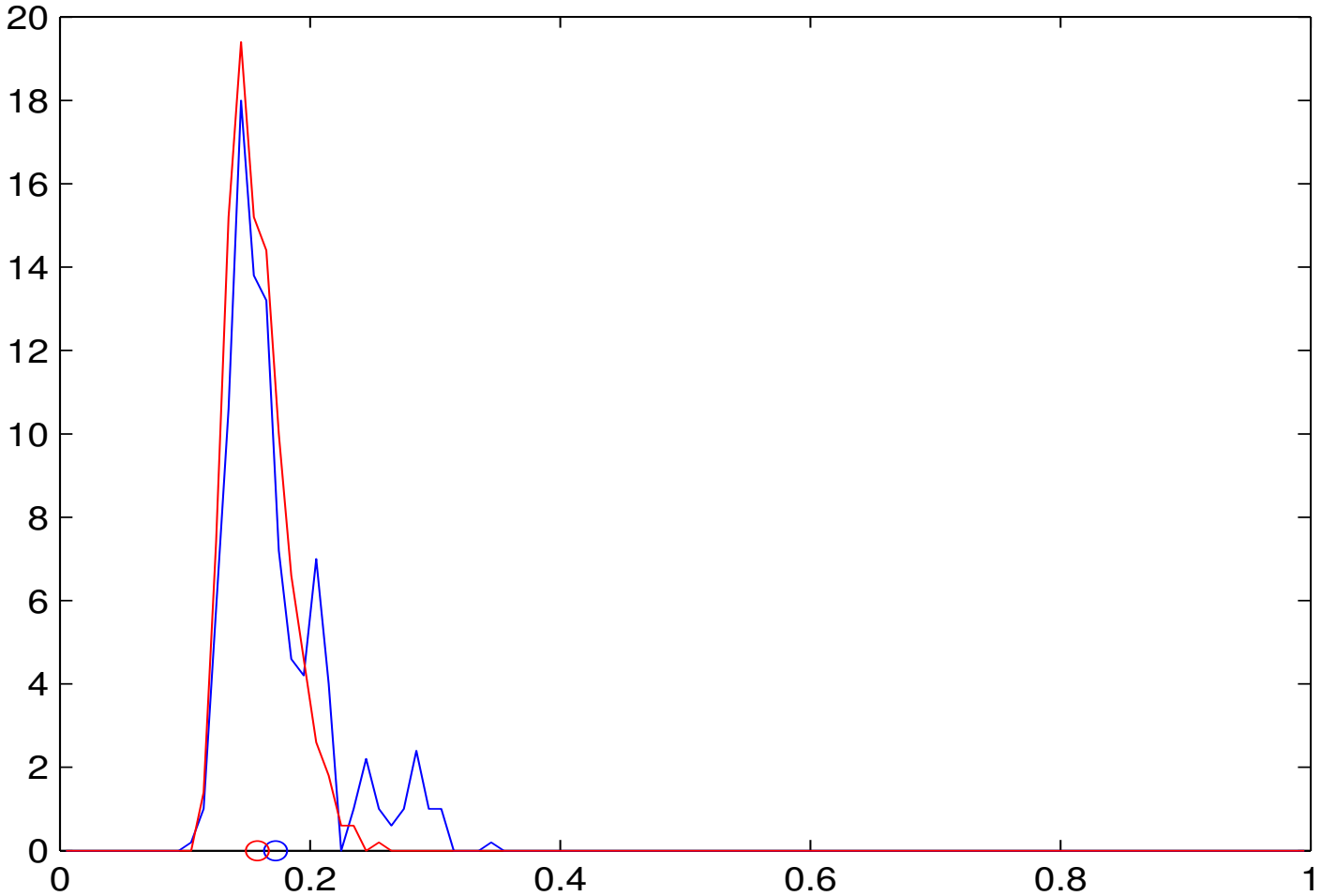
Error distribution: dataset size: 137



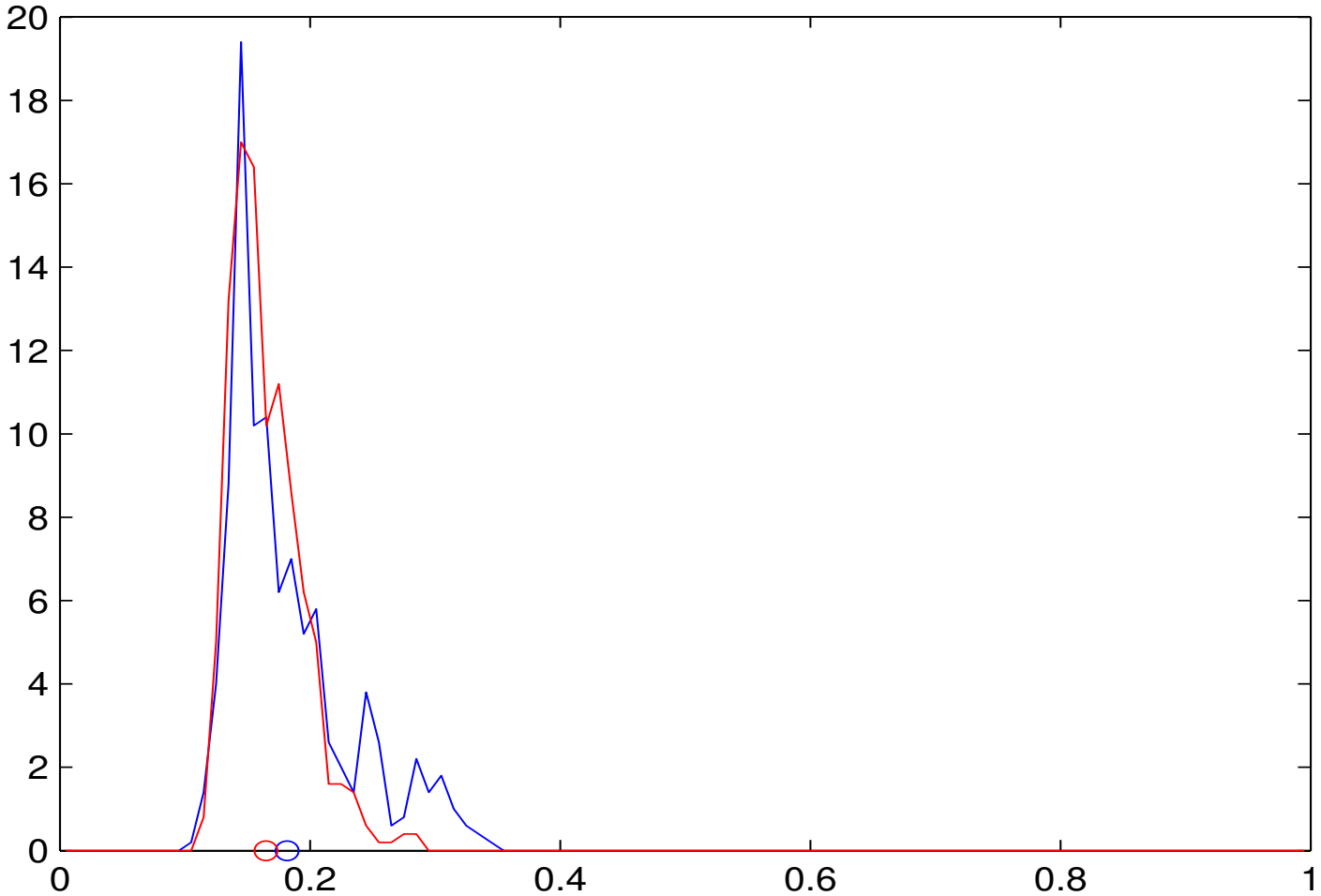
Error distribution: dataset size: 68



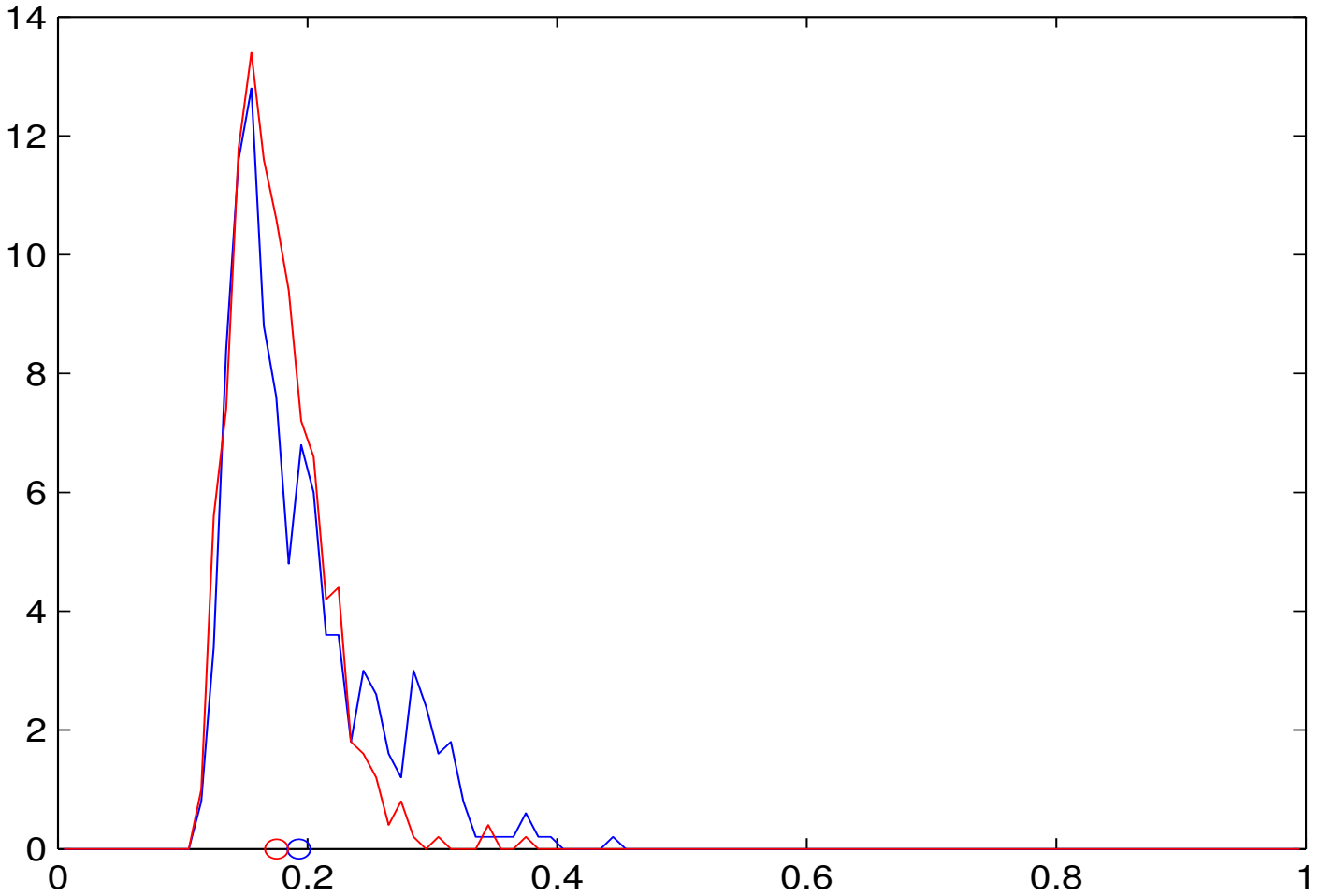
Error distribution: dataset size: 34



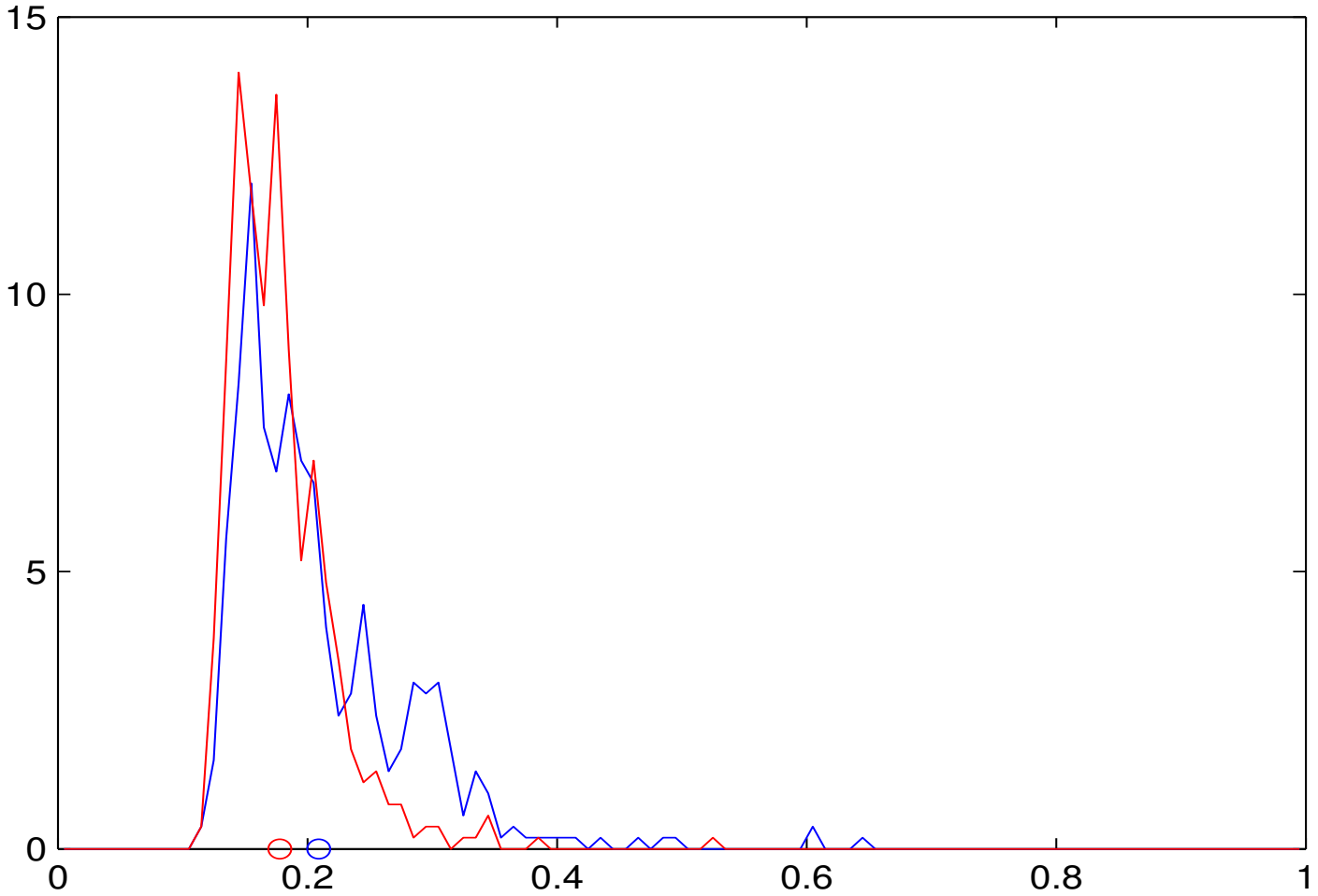
Error distribution: dataset size: 27



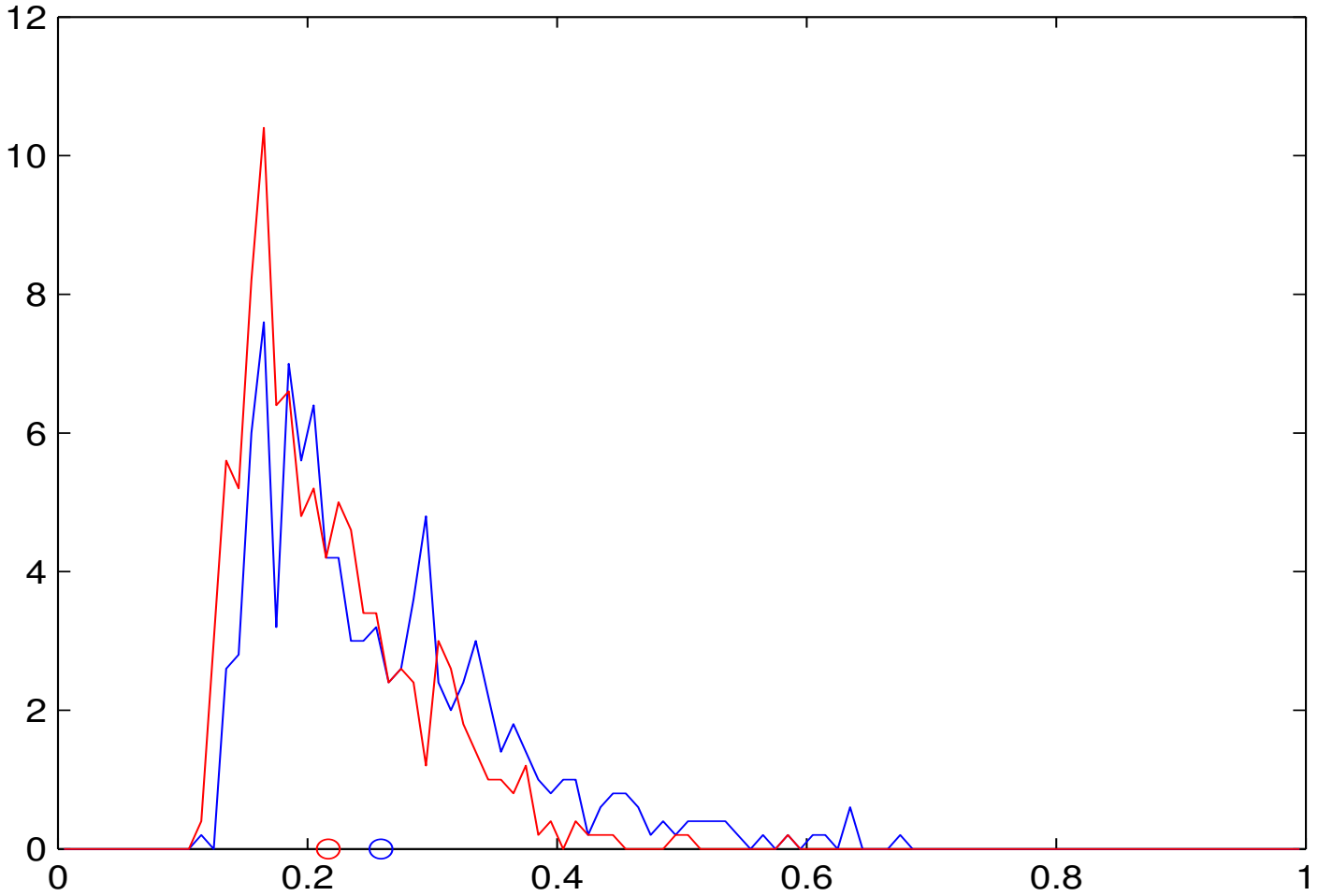
Error distribution: dataset size: 20



Error distribution: dataset size: 14



Error distribution: dataset size: 7



Dual form of the SVM problem

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

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

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

Dual form of the SVM problem

Taking derivatives gives:

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

Dual form of the SVM problem

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

which, again optimising with respect to λ, gives

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

Dual form of the SVM problem

equivalent to maximising

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

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

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:

$$\begin{aligned}
\text{choose} \quad & i, j \text{ such that } -C < \alpha_i^* y_i < 0 < \alpha_j^* y_j < C \\
b^* &= -0.5 \left(\sum_{k=1}^{m} \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_i) + \sum_{k=1}^{m} \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_j) \right) \\
f(\cdot) &= \operatorname{sgn} \left(\sum_{j=1}^{m} \alpha_j^* y_j \kappa(\mathbf{x}_j, \cdot) + b^* \right);
\end{aligned}$$

Dual form of the SVM problem

We can compute the margin as follows:

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

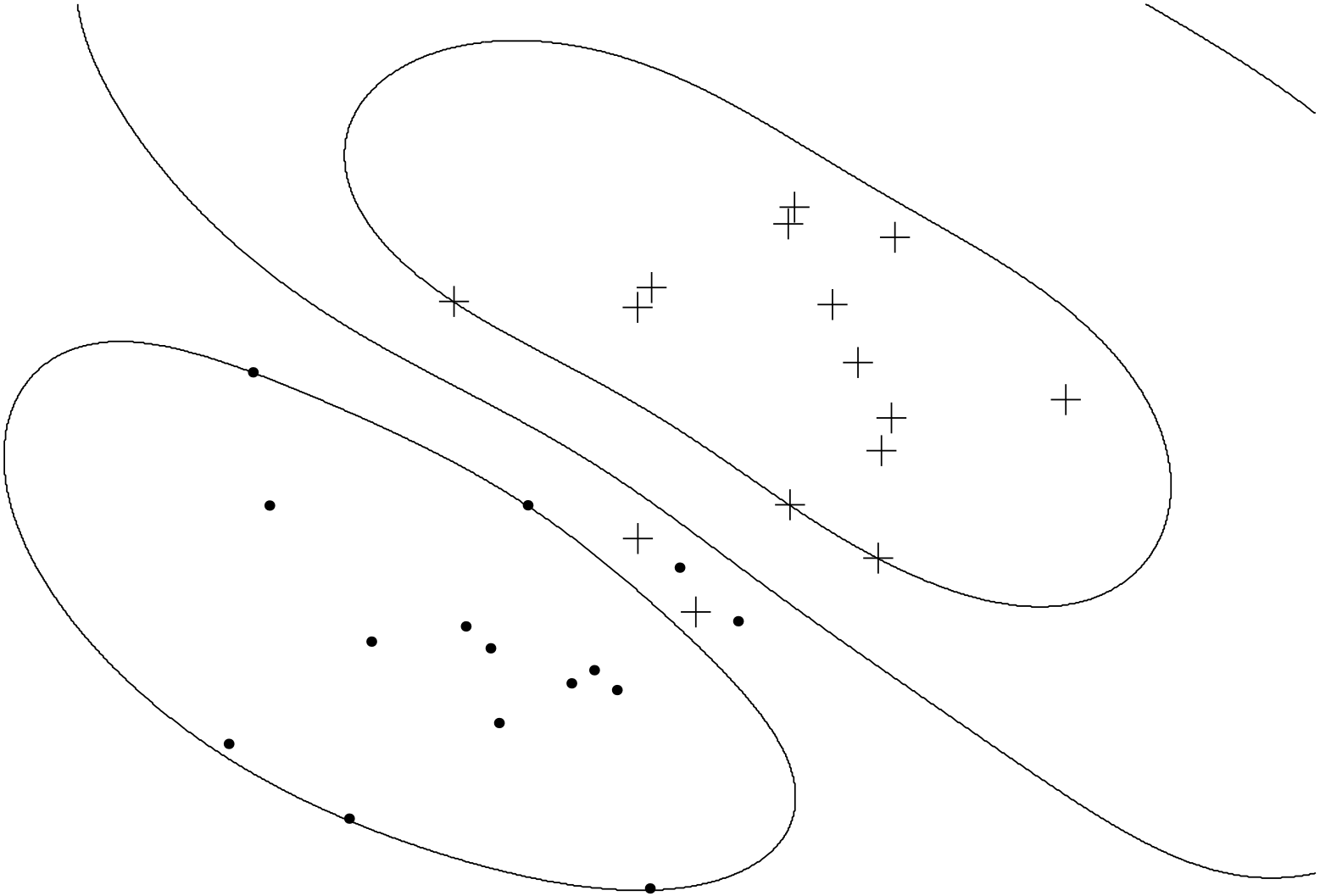
Similarly we can compute

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

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.

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.

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:

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

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.

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.

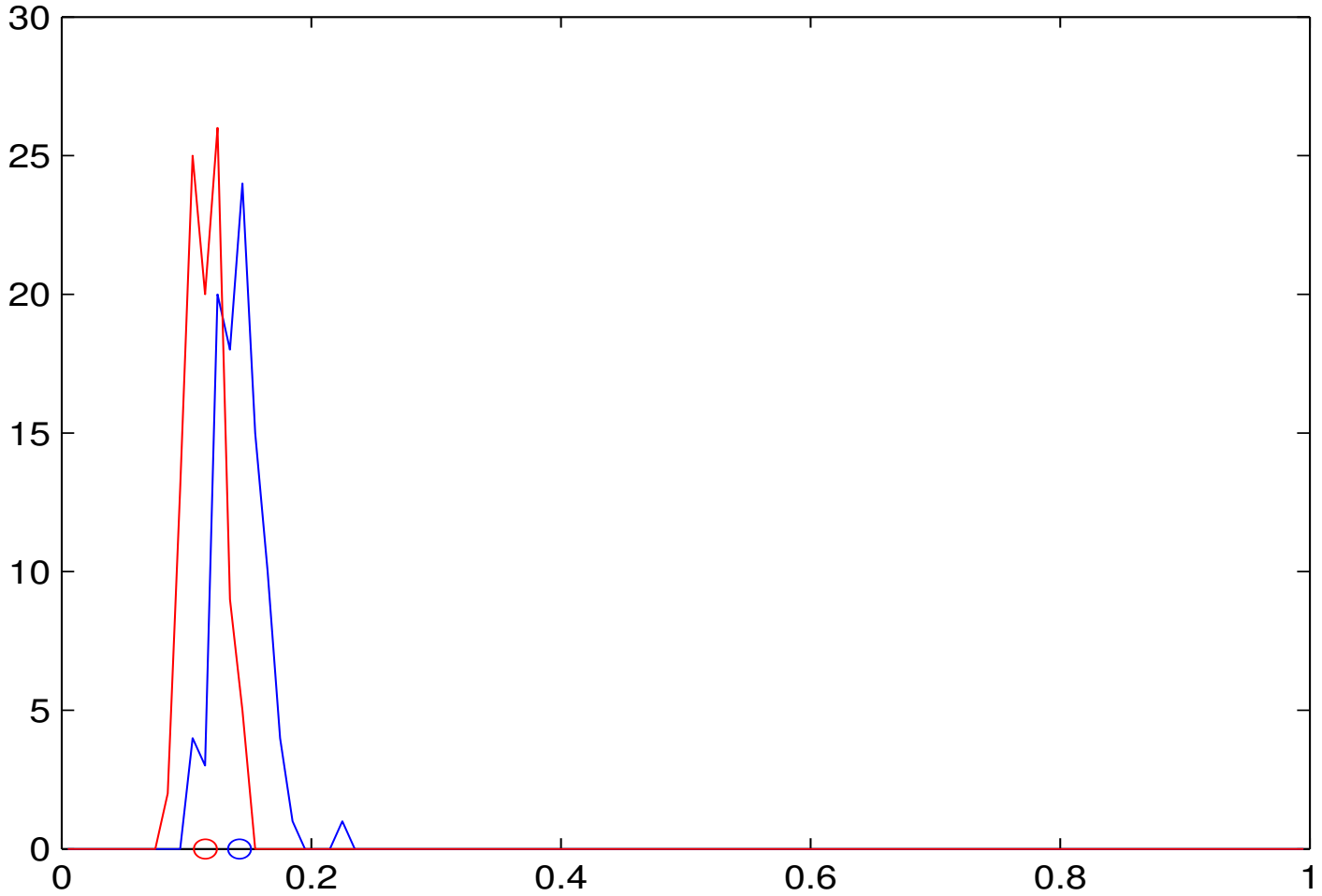
Alternative form of the SVM problem

- 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.

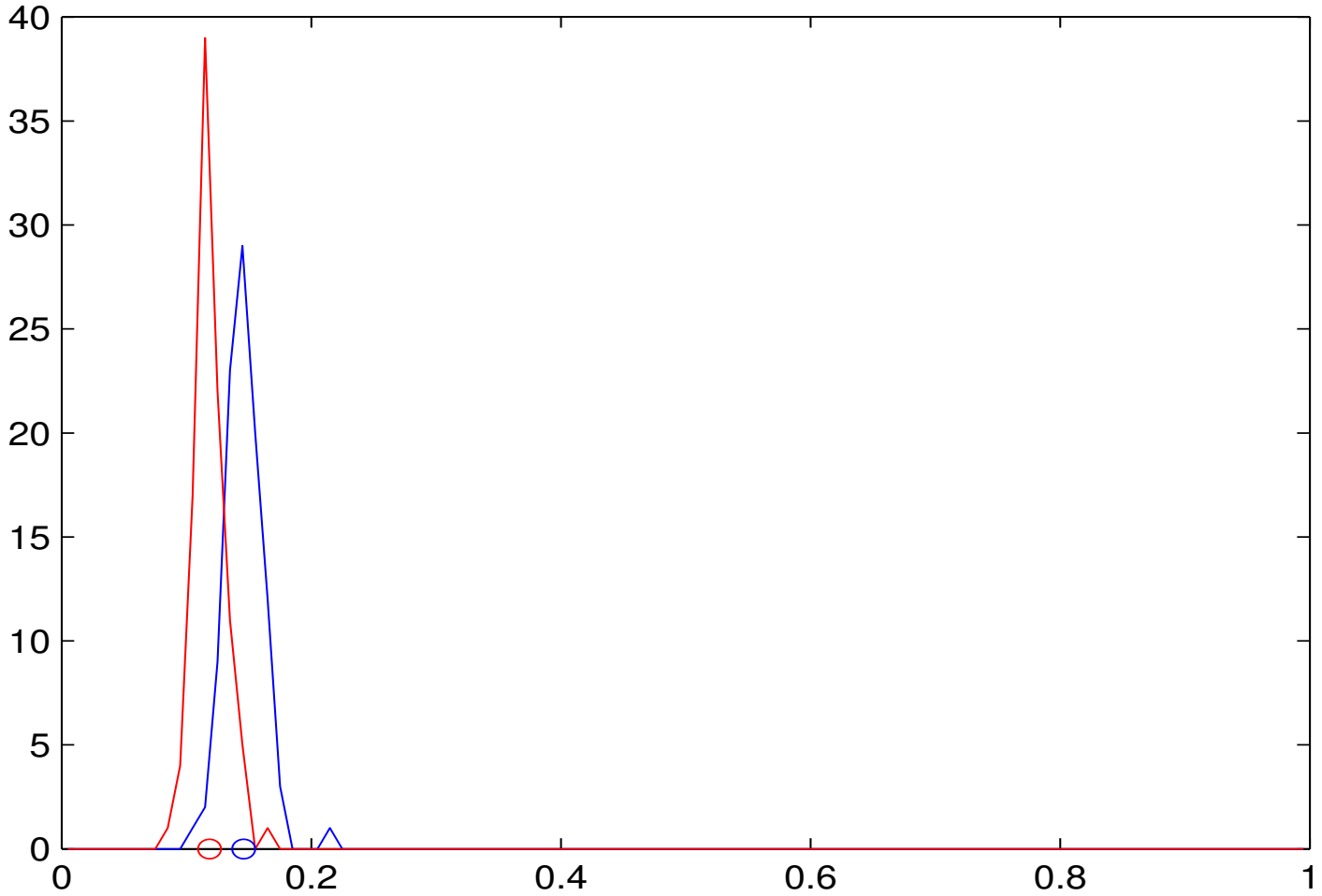
Alternative form of the SVM problem

- 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.

Error distribution: dataset size: 342



Error distribution: dataset size: 273



Linear programming machine

- Boosting leverages a large class H of 'weak learners' only able to classify slightly above 50% accuracy to build a (small) linear combination that performs very accurately.
- Some questions about why it works so well
- Seeks linear function in a feature space defined explicitly and can use the 1-norm to keep most coefficients zero.

Linear programming machine

- Boosting leverages a large class H of 'weak learners' only able to classify slightly above 50% accuracy to build a (small) linear combination that performs very accurately.
- Some questions about why it works so well
- Seeks linear function in a feature space defined explicitly and can use the 1-norm to keep most coefficients zero.

Linear programming machine

- Boosting leverages a large class H of 'weak learners' only able to classify slightly above 50% accuracy to build a (small) linear combination that performs very accurately.
- Some questions about why it works so well
- Seeks linear function in a feature space defined explicitly and can use the 1-norm to keep most coefficients zero.

Linear programming boosting

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

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

where $\mathbf{a}$ is the vector of coefficients.

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 $\xi_i = (1 - y_i \sum_h a_h h(\mathbf{x}_i))_+$ and $a_h \geq 0$.

- with corresponding bound:

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

where moving to a linear combination of the weak learners H has only cost factor of the 1-norm of the coefficients $(a_h)_{h \in H}$.

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 $\xi_i = (1 - y_i \sum_h a_h h(\mathbf{x}_i))_+$ and $a_h \geq 0$.

- with corresponding bound:

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

where moving to a linear combination of the weak learners H has only cost factor of the 1-norm of the coefficients $(a_h)_{h\in H}$.

Alternative version

- Can explicitly optimise margin with 1-norm fixed:

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

- Dual has the following form:

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

Alternative version

- Can explicitly optimise margin with 1-norm fixed:

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

- Dual has the following form:

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

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, β
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 criterion

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, β
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 criterion

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, β
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 criterion

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, β
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 criterion

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, β
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 criterion

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, β
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 criterion

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,

- equivalent to performing Linear Programming boosting over the (infinite) set of functions

$$\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t$$

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

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,

- equivalent to performing Linear Programming boosting over the (infinite) set of functions

$$\mathcal{F} = \bigcup_{t=1}^{N} \mathcal{F}_t$$

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

Multiple kernel learning

- The corresponding Rademacher bound gives

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

- provided we can bound

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

Multiple kernel learning

- The corresponding Rademacher bound gives

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

- provided we can bound

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

Bounding MKL

- First 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 $t = 0, \ldots, N$

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

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

$$\leq \frac{2}{m} \max_{1 \leq t \leq N} \sup_{f \in \mathcal{F}_t} \sum_{i=1}^{m} \sigma_i^* f(\mathbf{x}_i) + 4\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}}$$

$$\leq \frac{2}{m} \max_{1 \leq t \leq N} \hat{R}_m(\mathcal{F}_t) + 8\sqrt{\frac{\ln(2(N+1)/\delta)}{2m}}$$

with probability $1 - \delta/2$.

Bounding MKL

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

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

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

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

Bounding MKL

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

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

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

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

Experimental results with large-scale MKL

- Vedaldi et al. have applied to the PASCAL Visual Objects Challenge (VOC 2007) data and
  - improvements over the winners of the challenge in 17 out of the 20 categories
  - in more than half of the categories the increase in average precision was over 25%
  - have also scaled effectively to millions of kernels

$\star$ A. Vedaldi, V. Gulshan, M. Varma and A. Zisserman. Multiple kernels for object detection. In Proceedings CVPR, Kyoto, Japan, September 2009.

Linear Programming MKL

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

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

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.

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.

Frontiers of Machine Learning

- Deep learning has (re-)emerged as having important research and commercial value
- Deep belief networks and related approaches have led this charge
- Kernels are sometimes referred to as 'shallow'
- Now consider:
  - ways in which kernel approaches to learning has been made 'deeper'
  - possible integration of kernel and deep methods

Frontiers of Machine Learning

- Deep learning has (re-)emerged as having important research and commercial value
- Deep belief networks and related approaches have led this charge
- Kernels are sometimes referred to as 'shallow'
- Now consider:
  - ways in which kernel approaches to learning has been made 'deeper'
  - possible integration of kernel and deep methods

Frontiers of Machine Learning

- Deep learning has (re-)emerged as having important research and commercial value
- Deep belief networks and related approaches have led this charge
- Kernels are sometimes referred to as 'shallow'
- Now consider:
  - ways in which kernel approaches to learning has been made 'deeper'
  - possible integration of kernel and deep methods

Frontiers of Machine Learning

- Deep learning has (re-)emerged as having important research and commercial value
- Deep belief networks and related approaches have led this charge
- Kernels are sometimes referred to as 'shallow'
- Now consider:
  - ways in which kernel approaches to learning has been made 'deeper'
  - possible integration of kernel and deep methods

Why Shallow Learning?

- Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems
- However, they actually implement linear functions in the kernel defined feature space:

$$\mathbf{x} \longmapsto_{\text{fixed}} \phi(\mathbf{x}) \longmapsto_{\text{learned}} \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

  so that the learning (of $\mathbf{w}$) only occurs in one 'layer'.
- This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions
  - Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed
  - Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel

Why Shallow Learning?

- Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems
- However, they actually implement linear functions in the kernel defined feature space:

$$\mathbf{x} \longmapsto_{\text{fixed}} \phi(\mathbf{x}) \longmapsto_{\text{learned}} \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

  so that the learning (of $\mathbf{w}$) only occurs in one 'layer'.
- This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions
  - Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed
  - Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel

Why Shallow Learning?

- Kernels learn non-linear functions in the input space so would appear to be as flexible as deep learning systems
- However, they actually implement linear functions in the kernel defined feature space:

$$\mathbf{x} \longmapsto_{\text{fixed}} \phi(\mathbf{x}) \longmapsto_{\text{learned}} \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

  so that the learning (of $\mathbf{w}$) only occurs in one 'layer'.
- This is contrasted with deep learning where parameters are spread across several layers typically with non-linear transfer functions
  - Learning of the deeper layers is often unsupervised with the final classifier trained with the earlier layers fixed
  - Hence, we are effectively pre-learning a representation – this would be analogous to learning the kernel

What happens in practice?

- In practice we typically do perform some learning of the kernel:
  - fix some hyper-parameters via some heuristic (e.g. width σ of a Gaussian kernel)
  - use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)
- In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:
  - standard generalisation bounds no longer apply if we choose the feature space based on the training data
  - even test set bounds will be invalidated if we include the testing data in the representation learning phase
- Often more sophisticated representations encode 'deep' prior knowledge, but are 'learned' by trial and error
  - for example the histograms of patch cluster presence used in an object detection system

What happens in practice?

- In practice we typically do perform some learning of the kernel:
  - fix some hyper-parameters via some heuristic (e.g. width σ of a Gaussian kernel)
  - use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)
- In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:
  - standard generalisation bounds no longer apply if we choose the feature space based on the training data
  - even test set bounds will be invalidated if we include the testing data in the representation learning phase
- Often more sophisticated representations encode 'deep' prior knowledge, but are 'learned' by trial and error
  - for example the histograms of patch cluster presence used in an object detection system

What happens in practice?

- In practice we typically do perform some learning of the kernel:
  - fix some hyper-parameters via some heuristic (e.g. width σ of a Gaussian kernel)
  - use cross-validation to adapt the hyperparameter to optimise performance of the task (classification, regression, etc)
- In some respects this undermines the more principled approach espoused by kernel methods based on generalisation bounds:
  - standard generalisation bounds no longer apply if we choose the feature space based on the training data
  - even test set bounds will be invalidated if we include the testing data in the representation learning phase
- Often more sophisticated representations encode 'deep' prior knowledge, but are 'learned' by trial and error
  - for example the histograms of patch cluster presence used in an object detection system

Kernel defined features spaces

- Kernels define a feature space implicitly via a function $\kappa(\cdot,\cdot)$ that computes an inner product:
  - 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 are positive semi-definite:

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

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Implies that we can define kernels on virtually any objects provided these properties are satisfied, eg documents, graphs, networks, etc.
- Many standard linear algorithms can be implemented in the kernel defined feature space using a dual representation of the weight vectors – typically requires regularisation of the 2-norm
- Examples: ridge regression, PCA, CCA, SVMs, etc.

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Implies that we can define kernels on virtually any objects provided these properties are satisfied, eg documents, graphs, networks, etc.
- Many standard linear algorithms can be implemented in the kernel defined feature space using a dual representation of the weight vectors – typically requires regularisation of the 2-norm
- Examples: ridge regression, PCA, CCA, SVMs, etc.

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Implies that we can define kernels on virtually any objects provided these properties are satisfied, eg documents, graphs, networks, etc.
- Many standard linear algorithms can be implemented in the kernel defined feature space using a dual representation of the weight vectors – typically requires regularisation of the 2-norm
- Examples: ridge regression, PCA, CCA, SVMs, etc.

Kernel functions

- These two properties are all that is required for a kernel function to be valid: symmetric and every kernel matrix is psd.
- Implies that we can define kernels on virtually any objects provided these properties are satisfied, eg documents, graphs, networks, etc.
- Many standard linear algorithms can be implemented in the kernel defined feature space using a dual representation of the weight vectors – typically requires regularisation of the 2-norm
- Examples: ridge regression, PCA, CCA, SVMs, etc.

Generating the feature space

Proof outline:

- Define feature space as class of functions:

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

- Linear space with inner product defined by $\langle \kappa(\mathbf{x}, \cdot), \kappa(\mathbf{z}, \cdot) \rangle = \kappa(\mathbf{x}, \mathbf{z})$
- embedding given by

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

- Note reproducing property: for a function $f \in \mathcal{F}$

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

so called *Reproducing Kernel Hilbert Space (RKHS)*

Generating the feature space

Proof outline:

- Define feature space as class of functions:

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

- Linear space with inner product defined by $\langle \kappa(\mathbf{x}, \cdot), \kappa(\mathbf{z}, \cdot) \rangle = \kappa(\mathbf{x}, \mathbf{z})$
- embedding given by

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

- Note reproducing property: for a function $f \in \mathcal{F}$

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

so called *Reproducing Kernel Hilbert Space (RKHS)*

Generating the feature space

Proof outline:

- Define feature space as class of functions:

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

- Linear space with inner product defined by $\langle \kappa(\mathbf{x}, \cdot), \kappa(\mathbf{z}, \cdot) \rangle = \kappa(\mathbf{x}, \mathbf{z})$
- embedding given by

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

- Note reproducing property: for a function $f \in \mathcal{F}$

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

so called *Reproducing Kernel Hilbert Space (RKHS)*

Generating the feature space

Proof outline:

- Define feature space as class of functions:

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

- Linear space with inner product defined by $\langle \kappa(\mathbf{x}, \cdot), \kappa(\mathbf{z}, \cdot) \rangle = \kappa(\mathbf{x}, \mathbf{z})$
- embedding given by

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

- Note reproducing property: for a function $f \in \mathcal{F}$

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

so called *Reproducing Kernel Hilbert Space (RKHS)*

Mean Embeddings

- If there is a distribution $\mathcal{D}$ on the input space it defines a point $\mu_{\mathcal{D}}$ in the feature space:

$$\mu_{\mathcal{D}} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\phi(\mathbf{x})]$$

- and its empirical counterpart for a finite sample S:

$$\hat{\mu}_S = \mathbb{E}_{\mathbf{x}\sim S}[\phi(\mathbf{x})] = \hat{\mathbb{E}}[\phi(\mathbf{x})]$$

- Surprisingly, despite their being no restriction on the dimensionality of $\mathcal{F}$, if $\mathcal{D}$ has support in the R ball, with probability at least $1-\delta$ over an iid sample S

$$\|\mu_{\mathcal{D}} - \hat{\mu}_S\| \leq \frac{R}{\sqrt{m}}\left(2 + \sqrt{2\ln\frac{1}{\delta}}\right)$$

Mean Embeddings

- If there is a distribution $\mathcal{D}$ on the input space it defines a point $\mu_{\mathcal{D}}$ in the feature space:

$$\mu_{\mathcal{D}} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\phi(\mathbf{x})]$$

- and its empirical counterpart for a finite sample S:

$$\hat{\mu}_S = \mathbb{E}_{\mathbf{x}\sim S}[\phi(\mathbf{x})] = \hat{\mathbb{E}}[\phi(\mathbf{x})]$$

- Surprisingly, despite their being no restriction on the dimensionality of $\mathcal{F}$, if $\mathcal{D}$ has support in the R ball, with probability at least $1-\delta$ over an iid sample S

$$\|\mu_{\mathcal{D}} - \hat{\mu}_S\| \leq \frac{R}{\sqrt{m}}\left(2 + \sqrt{2\ln\frac{1}{\delta}}\right)$$

Mean Embeddings

- If there is a distribution $\mathcal{D}$ on the input space it defines a point $\mu_{\mathcal{D}}$ in the feature space:

$$\mu_{\mathcal{D}} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\phi(\mathbf{x})]$$

- and its empirical counterpart for a finite sample S:

$$\hat{\mu}_S = \mathbb{E}_{\mathbf{x}\sim S}[\phi(\mathbf{x})] = \hat{\mathbb{E}}[\phi(\mathbf{x})]$$

- Surprisingly, despite their being no restriction on the dimensionality of $\mathcal{F}$, if $\mathcal{D}$ has support in the R ball, with probability at least $1-\delta$ over an iid sample S

$$\|\mu_{\mathcal{D}} - \hat{\mu}_S\| \leq \frac{R}{\sqrt{m}}\left(2 + \sqrt{2\ln\frac{1}{\delta}}\right)$$

Mean Embeddings for Expectations

- Since $\mathcal{F}$ is a space of functions, using the reproducing property we can also estimate expectations for $f \in \mathcal{F}$

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\langle f, \phi(\mathbf{x})\rangle] = \langle f, \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\phi(\mathbf{x})]\rangle = \langle f, \mu_{\mathcal{D}}\rangle$$

- similarly for its empirical counterpart and furthermore we can bound the error of the empirical estimate:

$$\begin{aligned} |\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] - \hat{\mathbb{E}}[f(\mathbf{x})]| &= |\langle f, \mu_{\mathcal{D}}\rangle - \langle f, \hat{\mu}_S\rangle| \\ &= |\langle f, \mu_{\mathcal{D}} - \hat{\mu}_S\rangle \\ &\leq \frac{\|f\|R}{\sqrt{m}}\left(2 + \sqrt{2\ln\frac{1}{\delta}}\right) \end{aligned}$$

Mean Embeddings for Expectations

- Since $\mathcal{F}$ is a space of functions, using the reproducing property we can also estimate expectations for $f \in \mathcal{F}$

$$\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\langle f, \phi(\mathbf{x})\rangle] = \langle f, \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\phi(\mathbf{x})]\rangle = \langle f, \mu_{\mathcal{D}}\rangle$$

- similarly for its empirical counterpart and furthermore we can bound the error of the empirical estimate:

$$\begin{aligned} |\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[f(\mathbf{x})] - \hat{\mathbb{E}}[f(\mathbf{x})]| &= |\langle f, \mu_{\mathcal{D}}\rangle - \langle f, \hat{\mu}_S\rangle| \\ &= |\langle f, \mu_{\mathcal{D}} - \hat{\mu}_S\rangle \\ &\leq \frac{\|f\|R}{\sqrt{m}}\left(2 + \sqrt{2\ln\frac{1}{\delta}}\right) \end{aligned}$$

Conditional Mean Embeddings

- It is natural to ask if we can generalise these ideas to conditional distributions $P(X|Y)$
- Now the distribution and hence mean embedding is a function of Y

$$\mu_{P(X|Y)} = \mu(Y) \in \mathcal{F}$$

- Hence $\mu : Y \longrightarrow \mathcal{F}$ can be viewed as a regressor, albeit into a Hilbert space.

Conditional Mean Embeddings

- It is natural to ask if we can generalise these ideas to conditional distributions $P(X|Y)$
- Now the distribution and hence mean embedding is a function of Y

$$\mu_{P(X|Y)} = \mu(Y) \in \mathcal{F}$$

- Hence $\mu : Y \longrightarrow \mathcal{F}$ can be viewed as a regressor, albeit into a Hilbert space.

Conditional Mean Embeddings

- It is natural to ask if we can generalise these ideas to conditional distributions $P(X|Y)$
- Now the distribution and hence mean embedding is a function of Y

$$\mu_{P(X|Y)} = \mu(Y) \in \mathcal{F}$$

- Hence $\mu : Y \longrightarrow \mathcal{F}$ can be viewed as a regressor, albeit into a Hilbert space.

Reinforcement Learning (RL)

- Reinforcement Learning models agents that learn through acting in an environment and receiving reward
- AKA Markov Decision Processes:
  - set of states $\mathcal{S}$
  - set of actions $\mathcal{A}$
  - Markov transition kernel $P(s'|s,a)$
  - reward function $r : \mathcal{S} \times \mathcal{A} \to [0,1])$
  - and discount $0 < \gamma < 1$
- Agent has to select a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$

Reinforcement Learning (RL)

- Reinforcement Learning models agents that learn through acting in an environment and receiving reward
- AKA Markov Decision Processes:
  - set of states $\mathcal{S}$
  - set of actions $\mathcal{A}$
  - Markov transition kernel $P(s'|s,a)$
  - reward function $r : \mathcal{S} \times \mathcal{A} \to [0,1])$
  - and discount $0 < \gamma < 1$
- Agent has to select a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$

Reinforcement Learning (RL)

- Reinforcement Learning models agents that learn through acting in an environment and receiving reward
- AKA Markov Decision Processes:
  - set of states $\mathcal{S}$
  - set of actions $\mathcal{A}$
  - Markov transition kernel $P(s'|s,a)$
  - reward function $r : \mathcal{S} \times \mathcal{A} \to [0,1])$
  - and discount $0 < \gamma < 1$
- Agent has to select a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$

Experience in RL

- Agent experience: $\xi = (S_1, A_1, S_2, A_2, \ldots)$
- $S_1 \sim P_1$, $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot|S_t, A_t)$
- Expected return:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|\pi\right]$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s, \pi\right]$$

- Action Value function:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}\left[V^{\pi}(S')\right]$$

Experience in RL

- Agent experience: $\xi = (S_1, A_1, S_2, A_2, \ldots)$
- $S_1 \sim P_1$, $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot|S_t, A_t)$
- Expected return:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|\pi\right]$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s, \pi\right]$$

- Action Value function:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}\left[V^{\pi}(S')\right]$$

Experience in RL

- Agent experience: $\xi = (S_1, A_1, S_2, A_2, \ldots)$
- $S_1 \sim P_1$, $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot|S_t, A_t)$
- Expected return:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|\pi\right]$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s, \pi\right]$$

- Action Value function:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}\left[V^{\pi}(S')\right]$$

Experience in RL

- Agent experience: $\xi = (S_1, A_1, S_2, A_2, \ldots)$
- $S_1 \sim P_1$, $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot|S_t, A_t)$
- Expected return:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|\pi\right]$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s, \pi\right]$$

- Action Value function:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}\left[V^{\pi}(S')\right]$$

Experience in RL

- Agent experience: $\xi = (S_1, A_1, S_2, A_2, \ldots)$
- $S_1 \sim P_1$, $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot|S_t, A_t)$
- Expected return:

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|\pi\right]$$

- Value function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(S_t, A_t)|S_1 = s, \pi\right]$$

- Action Value function:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,a)}\left[V^{\pi}(S')\right]$$

Bellman Equation

- V^π satisfies the Bellman equation

$$V^\pi(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V^\pi(S')]\right]$$

- T^π is the Bellman operator mapping V to $T^\pi V$

$$(T^\pi V)(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V(S')]\right]$$

- The optimal policy satisfies: $V^\star(s) = \sup_{\pi\in\Pi} V^\pi(s)$
- The optimal policy can be computed by value iteration, policy iteration or dynamic programming

Bellman Equation

- V^π satisfies the Bellman equation

$$V^\pi(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V^\pi(S')]\right]$$

- T^π is the Bellman operator mapping V to $T^\pi V$

$$(T^\pi V)(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V(S')]\right]$$

- The optimal policy satisfies: $V^\star(s) = \sup_{\pi\in\Pi} V^\pi(s)$
- The optimal policy can be computed by value iteration, policy iteration or dynamic programming

Bellman Equation

- V^π satisfies the Bellman equation

$$V^\pi(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V^\pi(S')]\right]$$

- T^π is the Bellman operator mapping V to $T^\pi V$

$$(T^\pi V)(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V(S')]\right]$$

- The optimal policy satisfies: $V^\star(s) = \sup_{\pi\in\Pi} V^\pi(s)$
- The optimal policy can be computed by value iteration, policy iteration or dynamic programming

Bellman Equation

- V^π satisfies the Bellman equation

$$V^\pi(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V^\pi(S')]\right]$$

- T^π is the Bellman operator mapping V to $T^\pi V$

$$(T^\pi V)(s) = \mathbb{E}_{A\sim\pi(s)}\left[r(s,A) + \gamma \mathbb{E}_{S'\sim P(\cdot|s,A)}[V(S')]\right]$$

- The optimal policy satisfies: $V^\star(s) = \sup_{\pi\in\Pi} V^\pi(s)$
- The optimal policy can be computed by value iteration, policy iteration or dynamic programming

Linear representations of the Value function

- Can assume a linear form for V^π:

$$V^\pi(s) = \langle w_\pi, \phi(s) \rangle$$

- then solving the Bellman equation means finding w_π such that

$$\langle w_\pi, \phi(s) \rangle = r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))} \left[\langle w_\pi, \phi(S') \rangle \right]$$

- Building a model of the dynamics makes it possible to solve this
- Using CMEs gives a clean way of computing the expectations

Linear representations of the Value function

- Can assume a linear form for V^π:

$$V^\pi(s) = \langle w_\pi, \phi(s) \rangle$$

- then solving the Bellman equation means finding w_π such that

$$\langle w_\pi, \phi(s) \rangle = r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))} \left[\langle w_\pi, \phi(S') \rangle \right]$$

- Building a model of the dynamics makes it possible to solve this
- Using CMEs gives a clean way of computing the expectations

Linear representations of the Value function

- Can assume a linear form for V^π:

$$V^\pi(s) = \langle w_\pi, \phi(s) \rangle$$

- then solving the Bellman equation means finding w_π such that

$$\langle w_\pi, \phi(s) \rangle = r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))} \left[\langle w_\pi, \phi(S') \rangle \right]$$

- Building a model of the dynamics makes it possible to solve this
- Using CMEs gives a clean way of computing the expectations

Linear representations of the Value function

- Can assume a linear form for V^π:

$$V^\pi(s) = \langle w_\pi, \phi(s) \rangle$$

- then solving the Bellman equation means finding w_π such that

$$\langle w_\pi, \phi(s) \rangle = r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim P(\cdot|s,\pi(s))} \left[\langle w_\pi, \phi(S') \rangle \right]$$

- Building a model of the dynamics makes it possible to solve this
- Using CMEs gives a clean way of computing the expectations

- Recall if $V(s) = \langle v, \phi(s) \rangle$ then

$$\mathbb{E}_{S' \sim P(\cdot|s,a)} \left[V(S') \right] = \left\langle v, \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right] \right\rangle$$

- Hence we need $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$

$$\mu(s,a) = \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right]$$

- We learn regressor $\hat{\mu}$ using training loss

$$\hat{\text{loss}}(\hat{\mu}) = \frac{1}{m} \sum_{i=1}^{m} \|\hat{\mu}(s_i, a_i) - \phi(s_i')\|^2$$

where training data is $\{(s_i, a_i, s_i') : i = 1, \ldots, m\}$

- Recall if $V(s) = \langle v, \phi(s) \rangle$ then

$$\mathbb{E}_{S' \sim P(\cdot|s,a)} \left[V(S') \right] = \left\langle v, \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right] \right\rangle$$

- Hence we need $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$

$$\mu(s,a) = \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right]$$

- We learn regressor $\hat{\mu}$ using training loss

$$\hat{\text{loss}}(\hat{\mu}) = \frac{1}{m} \sum_{i=1}^{m} \|\hat{\mu}(s_i, a_i) - \phi(s_i')\|^2$$

where training data is $\{(s_i, a_i, s_i') : i = 1, \ldots, m\}$

- Recall if $V(s) = \langle v, \phi(s) \rangle$ then

$$\mathbb{E}_{S' \sim P(\cdot|s,a)} \left[V(S') \right] = \left\langle v, \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right] \right\rangle$$

- Hence we need $\mu : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$

$$\mu(s,a) = \mathbb{E}_{S' \sim P(\cdot|s,a)} \left[\phi(S') \right]$$

- We learn regressor $\hat{\mu}$ using training loss

$$\hat{\text{loss}}(\hat{\mu}) = \frac{1}{m} \sum_{i=1}^{m} \|\hat{\mu}(s_i, a_i) - \phi(s_i')\|^2$$

where training data is $\{(s_i, a_i, s_i') : i = 1, \ldots, m\}$

- Solution has the form $\hat{\mu}(s,a) = \sum_{i=1}^{m} \alpha_i(s,a)\phi(s'_i)$
- Hence

$$\mathbb{E}_{\hat{\mu}}\left[V(S')\right] = \langle \hat{\mu}(s,a), v\rangle = \sum_{i=1}^{m} \alpha_i(s,a)V(s'_i) = \alpha(s,a)^T \mathbf{v}$$

- So Bellman equation becomes a linear system on the vector $\mathbf{v}$

$$\mathbf{v} = \mathbf{r} + \gamma A\mathbf{v}$$

where $\mathbf{r} = \mathbb{E}_{A\sim\pi(s_i)} r(s_i, A)$ and $A_{ij} = \mathbb{E}_{A\sim\pi(s_i)}[\alpha_j(s_i, A)]$

- so reduced to finite MDP
- Size of the MDP rapidly becomes infeasible - so now turn to methods to control the number of states

- Solution has the form $\hat{\mu}(s,a) = \sum_{i=1}^{m} \alpha_i(s,a)\phi(s_i')$
- Hence

$$\mathbb{E}_{\hat{\mu}}\left[V(S')\right] = \langle \hat{\mu}(s,a), v\rangle = \sum_{i=1}^{m} \alpha_i(s,a)V(s_i') = \alpha(s,a)^T \mathbf{v}$$

- So Bellman equation becomes a linear system on the vector $\mathbf{v}$

$$\mathbf{v} = \mathbf{r} + \gamma A\mathbf{v}$$

where $\mathbf{r} = \mathbb{E}_{A\sim\pi(s_i)} r(s_i, A)$ and $A_{ij} = \mathbb{E}_{A\sim\pi(s_i)}[\alpha_j(s_i, A)]$

- so reduced to finite MDP
- Size of the MDP rapidly becomes infeasible - so now turn to methods to control the number of states

- Solution has the form $\hat{\mu}(s,a) = \sum_{i=1}^{m} \alpha_i(s,a)\phi(s_i')$
- Hence

$$\mathbb{E}_{\hat{\mu}}\left[V(S')\right] = \langle \hat{\mu}(s,a), v\rangle = \sum_{i=1}^{m} \alpha_i(s,a)V(s_i') = \alpha(s,a)^T \mathbf{v}$$

- So Bellman equation becomes a linear system on the vector $\mathbf{v}$

$$\mathbf{v} = \mathbf{r} + \gamma A\mathbf{v}$$

where $\mathbf{r} = \mathbb{E}_{A\sim\pi(s_i)} r(s_i, A)$ and $A_{ij} = \mathbb{E}_{A\sim\pi(s_i)}[\alpha_j(s_i, A)]$

- so reduced to finite MDP
- Size of the MDP rapidly becomes infeasible - so now turn to methods to control the number of states

- Solution has the form $\hat{\mu}(s,a) = \sum_{i=1}^{m} \alpha_i(s,a)\phi(s'_i)$
- Hence

$$\mathbb{E}_{\hat{\mu}}\left[V(S')\right] = \langle \hat{\mu}(s,a), v\rangle = \sum_{i=1}^{m} \alpha_i(s,a)V(s'_i) = \alpha(s,a)^T \mathbf{v}$$

- So Bellman equation becomes a linear system on the vector $\mathbf{v}$

$$\mathbf{v} = \mathbf{r} + \gamma A\mathbf{v}$$

where $\mathbf{r} = \mathbb{E}_{A\sim\pi(s_i)} r(s_i, A)$ and $A_{ij} = \mathbb{E}_{A\sim\pi(s_i)}[\alpha_j(s_i, A)]$

- so reduced to finite MDP
- Size of the MDP rapidly becomes infeasible - so now turn to methods to control the number of states

- Solution has the form $\hat{\mu}(s,a) = \sum_{i=1}^{m} \alpha_i(s,a)\phi(s_i')$
- Hence

$$\mathbb{E}_{\hat{\mu}}\left[V(S')\right] = \langle \hat{\mu}(s,a), v\rangle = \sum_{i=1}^{m} \alpha_i(s,a)V(s_i') = \alpha(s,a)^T \mathbf{v}$$

- So Bellman equation becomes a linear system on the vector $\mathbf{v}$

$$\mathbf{v} = \mathbf{r} + \gamma A\mathbf{v}$$

where $\mathbf{r} = \mathbb{E}_{A\sim\pi(s_i)} r(s_i, A)$ and $A_{ij} = \mathbb{E}_{A\sim\pi(s_i)}[\alpha_j(s_i, A)]$

- so reduced to finite MDP
- Size of the MDP rapidly becomes infeasible - so now turn to methods to control the number of states

Greedy compression set

`augmentCompressionSet`$(\mathcal{C}, \delta, \mathcal{P})$

Input: Initial compression set $\mathcal{C} = c_1, ..., c_m$, candidates $\mathcal{P} = s'_1, ..., s'_n$, tolerance δ

for $j = 1, 2, ..., n$ **do**

if $\min_{\boldsymbol{b} \in \mathbb{R}^m, ||\boldsymbol{b}||_1 \leq 1} || \sum_{i=1}^m b_i \phi(c_i) - \phi(s'_j) ||_{\mathcal{F}} > \delta$ **then**

Augment compression set: $\mathcal{C} \leftarrow \mathcal{C} \cup s'_j$, $m \leftarrow m + 1$

end if

end for

- For learning the α coefficients:

$$\alpha_j(s, a) = \sum_{i=1}^{m} K((s, a), (s_i, a_i)) W_{ij}$$

we have a regression problem – again with high complexity – turn to matching pursuit

Greedy compression set

`augmentCompressionSet`$(\mathcal{C}, \delta, \mathcal{P})$

Input: Initial compression set $\mathcal{C} = c_1, ..., c_m$, candidates $\mathcal{P} = s'_1, ..., s'_n$, tolerance δ

for $j = 1, 2, ..., n$ **do**

if $\min_{\boldsymbol{b} \in \mathbb{R}^m, ||\boldsymbol{b}||_1 \leq 1} || \sum_{i=1}^m b_i \phi(c_i) - \phi(s'_j) ||_{\mathcal{F}} > \delta$ **then**

Augment compression set: $\mathcal{C} \leftarrow \mathcal{C} \cup s'_j$, $m \leftarrow m + 1$

end if

end for

- For learning the α coefficients:

$$\alpha_j(s, a) = \sum_{i=1}^m K((s, a), (s_i, a_i)) W_{ij}$$

we have a regression problem – again with high complexity – turn to matching pursuit

Matching pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace
- Allows different criteria for selection to be implemented in one framework, eg sparse PCA, classification, regression, canonical correlation analysis, etc. and all come with bounds

⋆ Hussain, Z., Shawe-Taylor, J., Hardoon, D.R. and Dhanjal, C (2011) Design and Generalization Analysis of Orthogonal Matching Pursuit Algorithms, IEEE Trans on Information Theory, 57, 5326–5341.

Matching pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace
- Allows different criteria for selection to be implemented in one framework, eg sparse PCA, classification, regression, canonical correlation analysis, etc. and all come with bounds

  ⋆ Hussain, Z., Shawe-Taylor, J., Hardoon, D.R. and Dhanjal, C (2011) Design and Generalization Analysis of Orthogonal Matching Pursuit Algorithms, IEEE Trans on Information Theory, 57, 5326–5341.

Matching pursuit

- Matching pursuit greedily chooses training examples that determine directions in feature space that are well-suited to some task and then deflates
- Analysis combining sparse reconstruction with generalisation error bounds gives first bounds on performance in learnt subspace
- Allows different criteria for selection to be implemented in one framework, eg sparse PCA, classification, regression, canonical correlation analysis, etc. and all come with bounds

⋆ Hussain, Z., Shawe-Taylor, J., Hardoon, D.R. and Dhanjal, C (2011) Design and Generalization Analysis of Orthogonal Matching Pursuit Algorithms, IEEE Trans on Information Theory, 57, 5326–5341.

Matching pursuit for KCCA

Require: two views $\mathbf{K}_x$, $\mathbf{K}_y$ and sparsity parameter $k > 0$.

1: initialise index vector $\mathbf{i} = [\,]$ and an all one vector $\mathbf{1}$.

2: **for** $i = 1$ to k **do**

3: set $\mathbf{i}_i$ to index of $\max_i \frac{\mathbf{K}_x[:,i]'\mathbf{K}_y[:,i]}{\sqrt{\mathbf{K}_x^2[i,i]\mathbf{K}_y^2[i,i]}}$

4: set $\boldsymbol{\tau}_x = \mathbf{K}_x[:,\mathbf{i}_i]$ and $\boldsymbol{\tau}_y = \mathbf{K}_y[:,\mathbf{i}_i]$ to deflate kernel matrices:

$$\mathbf{K}_x = \mathbf{K}_x - \frac{\boldsymbol{\tau}_x\left(\boldsymbol{\tau}_x'\mathbf{K}_x\right)}{\boldsymbol{\tau}_x'\boldsymbol{\tau}_x}$$

$$\mathbf{K}_y = \mathbf{K}_y - \frac{\boldsymbol{\tau}_y\left(\boldsymbol{\tau}_y'\mathbf{K}_y\right)}{\boldsymbol{\tau}_y'\boldsymbol{\tau}_y}$$

5: **end for**

6: solve KCCA on points indexed by final $\mathbf{i}$ to find $\tilde{\boldsymbol{\alpha}}_x$ and $\tilde{\boldsymbol{\alpha}}_y$ the duals of the projection vectors.

Matching pursuit bound plot

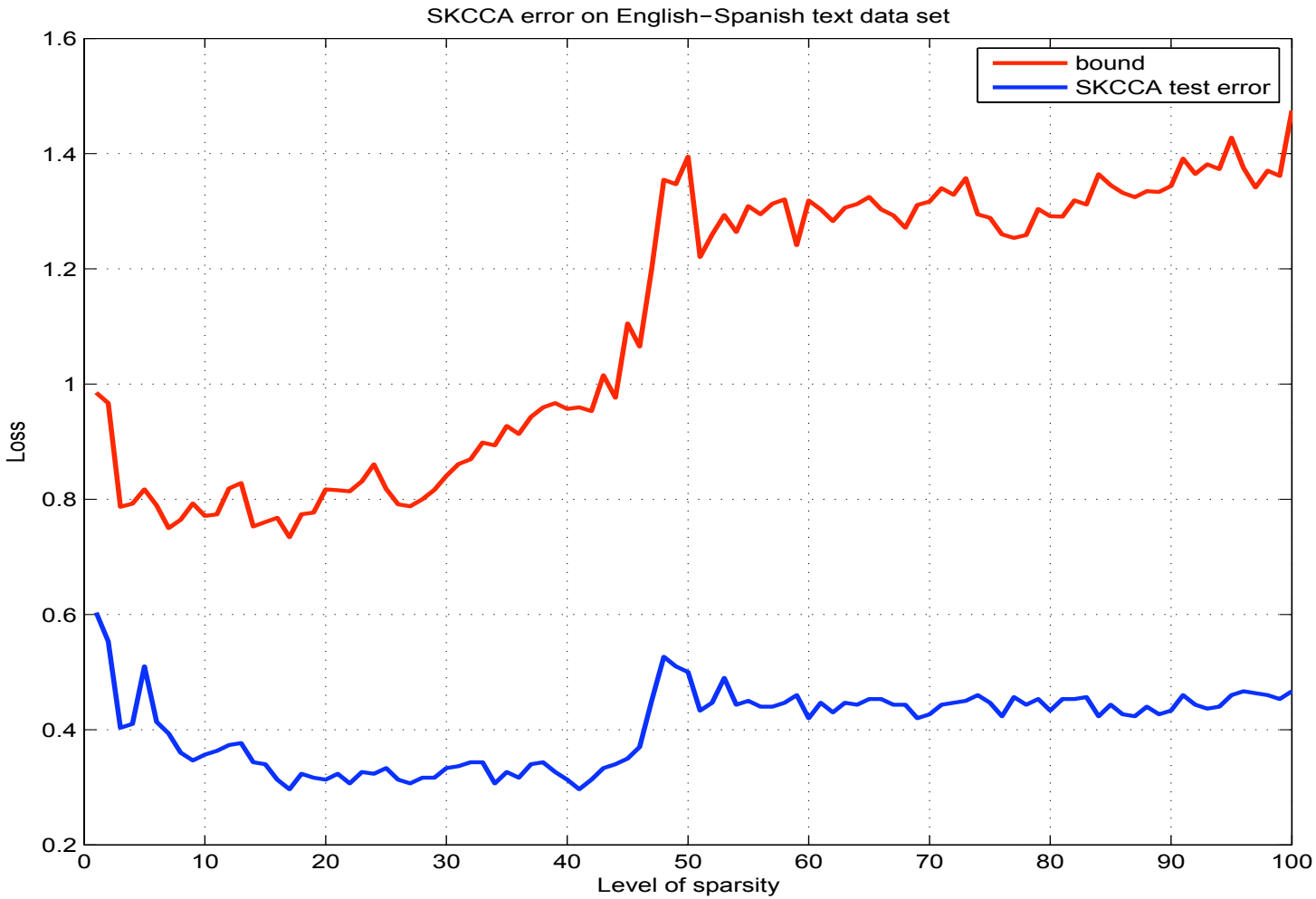


Figure: Bound plot for sparse KCCA using 1-dimension.

Policy iteration with Compressed kernel CMEs

for $k = 1, 2, \ldots$ **do**

$\mathcal{C}_k = \text{augmentCompressionSet}(\mathcal{C}_{k-1}, \{s'_i\}_{i=n_{k-1}+1}^{n_k}, \delta)$.

$\mathcal{G}_k = \mathcal{B}_{k-1} \cup \{K(\cdot, (s_i, a_i))\}_{i=n_{k-1}+1}^{n_k}$

Sparse basis selection: Learn sparse basis $\mathcal{B}_k = \{K(\cdot, (\hat{s}_\ell, \hat{a}_\ell))\}_{\ell=1}^{d_k}$, $d_k \leq d$ from candidates $\mathcal{G}_k$ using matching pursuit; Set $\psi_\ell^k(\cdot) = K((\hat{s}_\ell, \hat{a}_\ell), \cdot)$,

$\mathbf{\Psi}_k = \left(\psi^k(s_1, a_1), \ldots, \psi^k(s_{n_k}, a_{n_k})\right)^\top$.

$\mathbf{W}_k^{CMP} = (\mathbf{\Psi}_k^\top \mathbf{\Psi}_k + \lambda \mathbf{K})^{-1} \mathbf{\Psi}_k^\top \mathbf{L}^{DC} (\mathbf{L}^{CC})^{-1}$.

for $\ell = 1, 2, \ldots, J$ **do**

Policy evaluation: Using finite pseudo-MDP dynamics $\alpha^{PCMP}(c_j, a)$ for $a \in \mathcal{A}$, $c_j \in \mathcal{C}$ solve approximate Bellman Equation to obtain estimate $\hat{V}_\ell$ of V^{π_k} at the compression points $c_j \in \mathcal{C}$. Set

$\hat{Q}_\ell(s, a) = r(s, a) + \gamma \sum_{j=1}^{|\mathcal{C}_k|} \alpha_j^{PCMP}(s, a) \hat{V}_\ell(c_j)$.

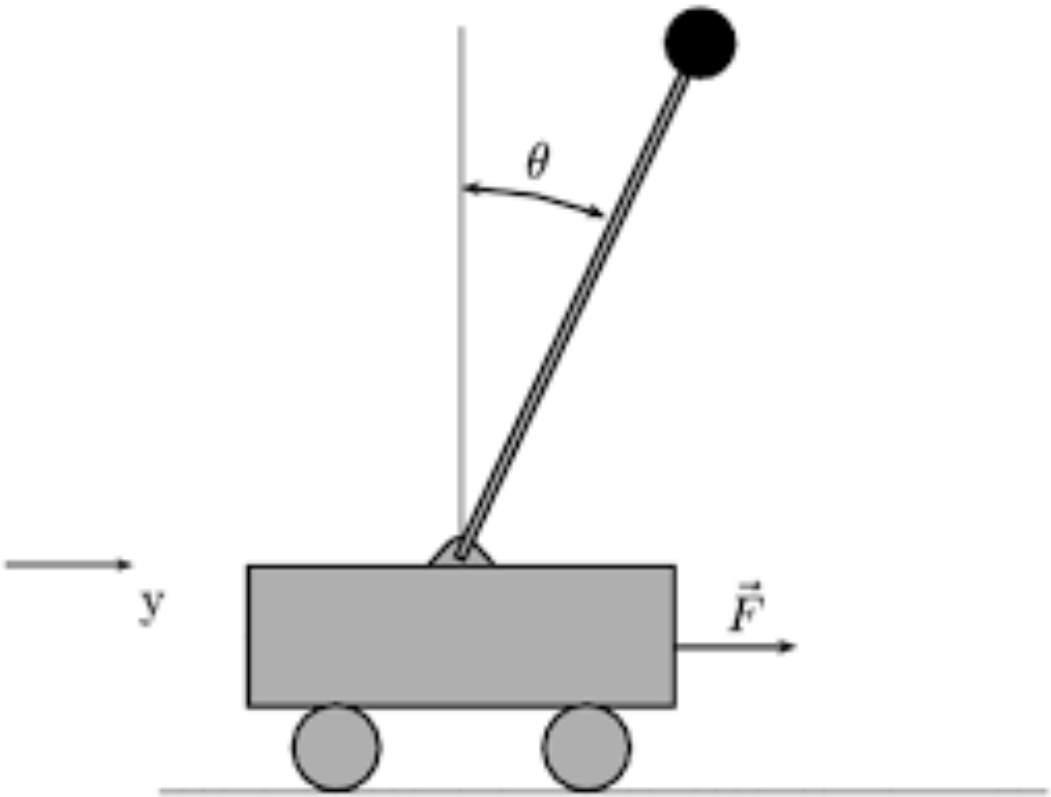
Policy improvement: $\pi_k \leftarrow \text{greedy}(\hat{Q}_\ell)$.

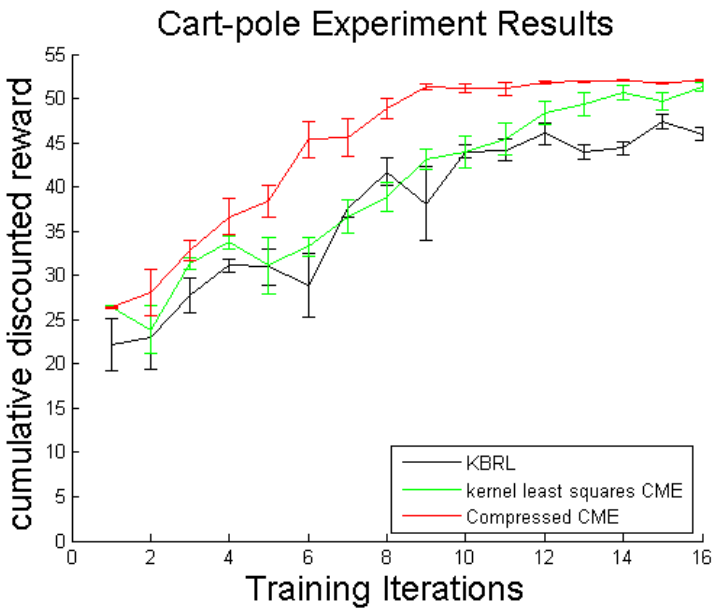
end for

Experiments: Cart-pole benchmark

Simulated under-actuated cart-pole swing-up benchmark problem

- $\mathcal{S} = \mathbb{R}^2$, $s = (\theta, \dot{\theta})$, $\mathcal{A} = [-50, 50]$, horizontal force in newtons





Experiments: Quadrocopter Simulator

Simulator calibrated to model the dynamics of Pelican^{TM} quadrocopter platforms
$\mathcal{S} \subset \mathbb{R}^{13}$, $s = (x, y, z, \theta\, \phi, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}, \dot{\phi}, \dot{\psi}, F)$
$\mathcal{A} \subset \mathbb{R}^3$ represents desired velocity vectors, PID controller translates into low level commands
Tasks:

- Navigation: platform must navigate to point
- Holding pattern: platform must stay in circle and maintain minimum velocity

Experiments: Quadrocopter Results

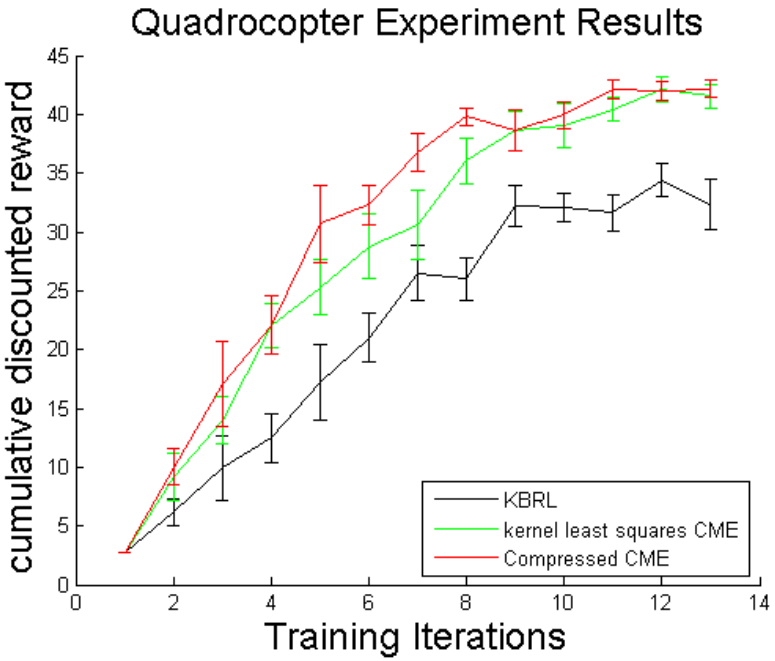


Figure: Quadrocopter tasks: navigation and holding pattern

RKHS controller **better in high-dim. state-space**

- Attempts to obtain similar results with deep learning have extended the flexibility and scaling of the method, albeit at the expense of requiring more training iterations.

Experiments: Quadrocopter Results

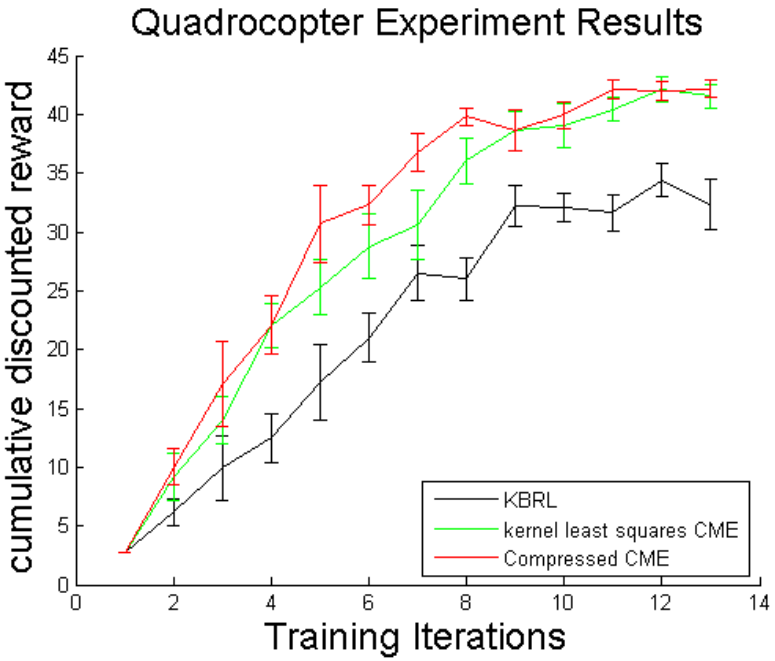


Figure: Quadrocopter tasks: navigation and holding pattern

RKHS controller **better in high-dim. state-space**

- Attempts to obtain similar results with deep learning have extended the flexibility and scaling of the method, albeit at the expense of requiring more training iterations.

Deep Learning models

- Deep learning models allow learning more complex models through multiple layers of parameters
- This means that the optimisation space is no longer convex and typically the problem is cast as seeking a local minimum through (stochastic) gradient descent
- Remarkably the generalisation performance does not seem to be adversely affected by using such flexible models
- Understanding what underpins this good performance is the subject of current theoretical studies

Deep Learning models

- Deep learning models allow learning more complex models through multiple layers of parameters
- This means that the optimisation space is no longer convex and typically the problem is cast as seeking a local minimum through (stochastic) gradient descent
- Remarkably the generalisation performance does not seem to be adversely affected by using such flexible models
- Understanding what underpins this good performance is the subject of current theoretical studies

Deep Learning models

- Deep learning models allow learning more complex models through multiple layers of parameters
- This means that the optimisation space is no longer convex and typically the problem is cast as seeking a local minimum through (stochastic) gradient descent
- Remarkably the generalisation performance does not seem to be adversely affected by using such flexible models
- Understanding what underpins this good performance is the subject of current theoretical studies

Deep Learning models

- Deep learning models allow learning more complex models through multiple layers of parameters
- This means that the optimisation space is no longer convex and typically the problem is cast as seeking a local minimum through (stochastic) gradient descent
- Remarkably the generalisation performance does not seem to be adversely affected by using such flexible models
- Understanding what underpins this good performance is the subject of current theoretical studies

Motivation

- Stochastic Gradient Descent uses mini-batches to derive noisy gradient estimates
- Simplest approach is average of gradients, but can include longer averages
- Conjugate gradient methods exploit second order information
- Can more be extracted from mini-batch gradients exploiting the fact that they represent an i.i.d. sample of the data distribution?

Motivation

- Stochastic Gradient Descent uses mini-batches to derive noisy gradient estimates
- Simplest approach is average of gradients, but can include longer averages
- Conjugate gradient methods exploit second order information
- Can more be extracted from mini-batch gradients exploiting the fact that they represent an i.i.d. sample of the data distribution?

Motivation

- Stochastic Gradient Descent uses mini-batches to derive noisy gradient estimates
- Simplest approach is average of gradients, but can include longer averages
- Conjugate gradient methods exploit second order information
- Can more be extracted from mini-batch gradients exploiting the fact that they represent an i.i.d. sample of the data distribution?

Motivation

- Stochastic Gradient Descent uses mini-batches to derive noisy gradient estimates
- Simplest approach is average of gradients, but can include longer averages
- Conjugate gradient methods exploit second order information
- Can more be extracted from mini-batch gradients exploiting the fact that they represent an i.i.d. sample of the data distribution?

Key idea

- View weight update direction as a classifier of the mini-batch: correct classification if reducing its error, incorrect if increasing its error
- This is ignoring second order effects: i.e. for small weight updates will hold

$$f^j(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}_i) \approx f^j(\mathbf{w}, \mathbf{x}_i) + \langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i) \rangle$$

- Have a target reduction of ϵ
- In order to minimise second order effects, we need to choose a minimum norm update that has the desired error reductions
- this can be translated into an optimisation that needs to be solved

Key idea

- View weight update direction as a classifier of the mini-batch: correct classification if reducing its error, incorrect if increasing its error
- This is ignoring second order effects: i.e. for small weight updates will hold

$$f^j(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}_i) \approx f^j(\mathbf{w}, \mathbf{x}_i) + \langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i) \rangle$$

- Have a target reduction of ϵ
- In order to minimise second order effects, we need to choose a minimum norm update that has the desired error reductions
- this can be translated into an optimisation that needs to be solved

Key idea

- View weight update direction as a classifier of the mini-batch: correct classification if reducing its error, incorrect if increasing its error
- This is ignoring second order effects: i.e. for small weight updates will hold

$$f^j(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}_i) \approx f^j(\mathbf{w}, \mathbf{x}_i) + \langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i) \rangle$$

- Have a target reduction of ϵ
- In order to minimise second order effects, we need to choose a minimum norm update that has the desired error reductions
- this can be translated into an optimisation that needs to be solved

Key idea

- View weight update direction as a classifier of the mini-batch: correct classification if reducing its error, incorrect if increasing its error
- This is ignoring second order effects: i.e. for small weight updates will hold

$$f^j(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}_i) \approx f^j(\mathbf{w}, \mathbf{x}_i) + \langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i) \rangle$$

- Have a target reduction of ϵ
- In order to minimise second order effects, we need to choose a minimum norm update that has the desired error reductions
- this can be translated into an optimisation that needs to be solved

Key idea

- View weight update direction as a classifier of the mini-batch: correct classification if reducing its error, incorrect if increasing its error
- This is ignoring second order effects: i.e. for small weight updates will hold

$$f^j(\mathbf{w} + \delta\mathbf{w}, \mathbf{x}_i) \approx f^j(\mathbf{w}, \mathbf{x}_i) + \langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i) \rangle$$

- Have a target reduction of ϵ
- In order to minimise second order effects, we need to choose a minimum norm update that has the desired error reductions
- this can be translated into an optimisation that needs to be solved

Update optimisation to choose $\delta\mathbf{w}$

1: Minimise $\frac{1}{2}\|\delta\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}\sum_{j=1}^{K}\xi_{ij}$
2: Subject to: $y_{ij}\langle\delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x}_i)\rangle \geq \epsilon - \xi_{ij}$ $\xi_{ij} \geq 0$, $i = 1, \ldots, \ell; j = 1, \ldots, K$

this is an SVM optimisation (with target margin ϵ).

The dual optimisation is

1: Max $\epsilon\sum_{ij}\alpha_{ij} - \frac{1}{2}\sum_{ijkl}\alpha_{ij}\alpha_{kl}\kappa((\mathbf{x}_i, j), (\mathbf{x}_k, l))$
2: Subject to: $C \geq \alpha_{ij} \geq 0$

where $\kappa((\mathbf{x}_i, j), (\mathbf{x}_k, l)) = \langle\nabla f^j(\mathbf{w}, \mathbf{x}_i), \nabla f^l(\mathbf{w}, \mathbf{x}_k)\rangle$

Analysis

- Since data generated iid, can use generalisation bounds to analyse the effects of the weight update
- The following bound holds

$$\mathbb{E}_{\mathcal{D}}[(\epsilon - y_j\langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x})\rangle)_+] \leq A = \frac{1}{\ell\epsilon}\sum_{ij}\xi_{ij} + \frac{4\|\delta\mathbf{w}\|}{\epsilon\ell}\sqrt{\mathrm{tr}(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}}$$

- Hence, ignoring second order effects, for an $\eta\delta\mathbf{w}$ weight update, the average hinge loss across the whole training (and test) set will with high probability reduce by at least

$$\eta\,(\epsilon - A)$$

Analysis

- Since data generated iid, can use generalisation bounds to analyse the effects of the weight update
- The following bound holds

$$\mathbb{E}_{\mathcal{D}}[(\epsilon - y_j\langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x})\rangle)_+] \leq A = \frac{1}{\ell\epsilon}\sum_{ij}\xi_{ij} + \\ + \frac{4\|\delta\mathbf{w}\|}{\epsilon\ell}\sqrt{\mathrm{tr}(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}}$$

- Hence, ignoring second order effects, for an $\eta\delta\mathbf{w}$ weight update, the average hinge loss across the whole training (and test) set will with high probability reduce by at least

$$\eta\,(\epsilon - A)$$

Analysis

- Since data generated iid, can use generalisation bounds to analyse the effects of the weight update
- The following bound holds

$$\mathbb{E}_{\mathcal{D}}[(\epsilon - y_j\langle \delta\mathbf{w}, \nabla f^j(\mathbf{w}, \mathbf{x})\rangle)_+] \leq A = \frac{1}{\ell\epsilon}\sum_{ij}\xi_{ij} + \\ + \frac{4\|\delta\mathbf{w}\|}{\epsilon\ell}\sqrt{\mathrm{tr}(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2\ell}}$$

- Hence, ignoring second order effects, for an $\eta\delta\mathbf{w}$ weight update, the average hinge loss across the whole training (and test) set will with high probability reduce by at least

$$\eta\,(\epsilon - A)$$

Optimization

$\eta = 0.1$; $r = 1.0$; $\ell =$ initial batch size
for $i \in [1, 2, \cdots, num\_iter]$ **do**
  $\mathcal{B}_s \leftarrow$ generateMinibatches($\mathcal{D}$, ℓ)
  $\delta \mathbf{w} \leftarrow$ trainMinibatch($\mathcal{B}_s$, $\mathcal{C}$, r)
  bound_term $= \|\delta \mathbf{w}\| \sqrt{tr(\mathbf{K})}/\ell$
  while bound_term $>$ threshold **do**
    $\ell \leftarrow 2 * \ell$ #minibatch size
    $r \leftarrow 0.1 * r$ #SVM Regularizer
    $\mathcal{B}_s \leftarrow$ generateMinibatches($\mathcal{D}$, ℓ)
    $\delta \mathbf{w} \leftarrow$ trainMinibatch($\mathcal{B}_s$, $\mathcal{C}$, r)
    bound_term $= \|\delta \mathbf{w}\| \sqrt{tr(\mathbf{K})}/\ell$
  end while
  $\mathbf{w} \leftarrow \mathbf{w} + \eta * \delta \mathbf{w}$
end for

Datasets

- MNIST: It consists of images of handwritten digits in binary. It has a training set of 60,000 examples, and a test set of 10,000 examples.
- CIFAR-10: It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.

Convergence

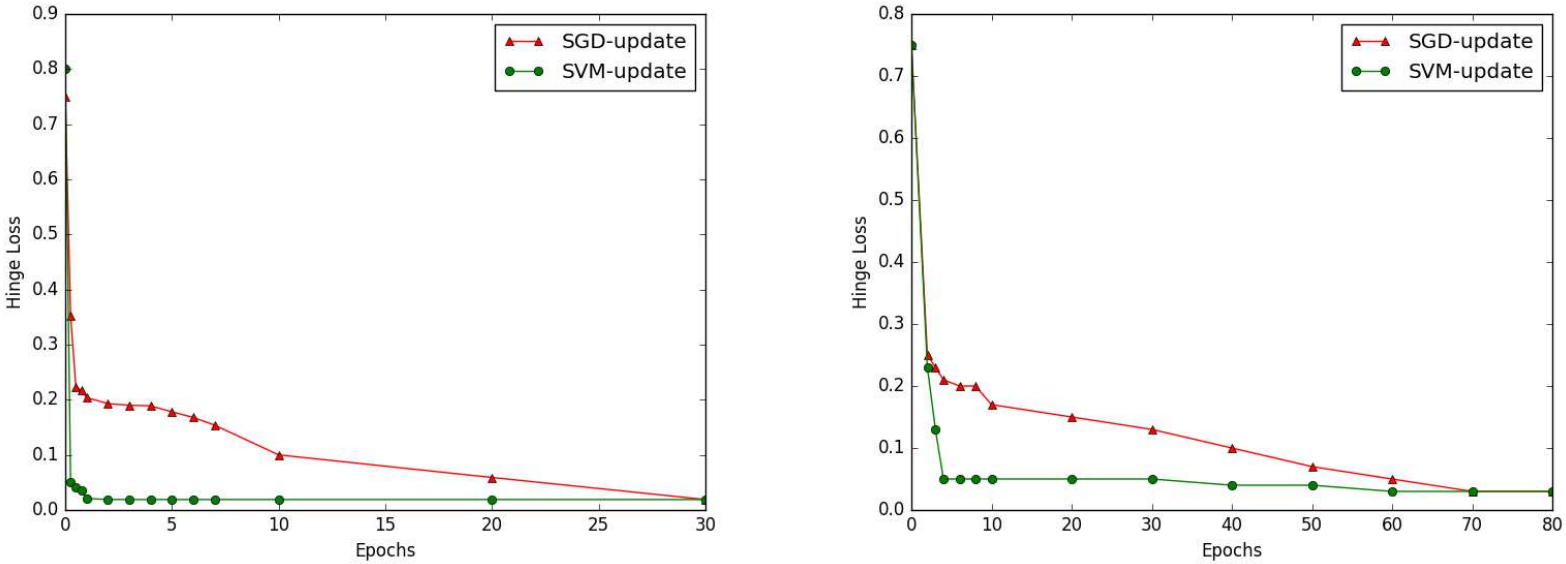


Figure: **Left**: MNIST & **Right**: CIFAR. We plot the hinge loss over train set versus epochs.

Accuracy

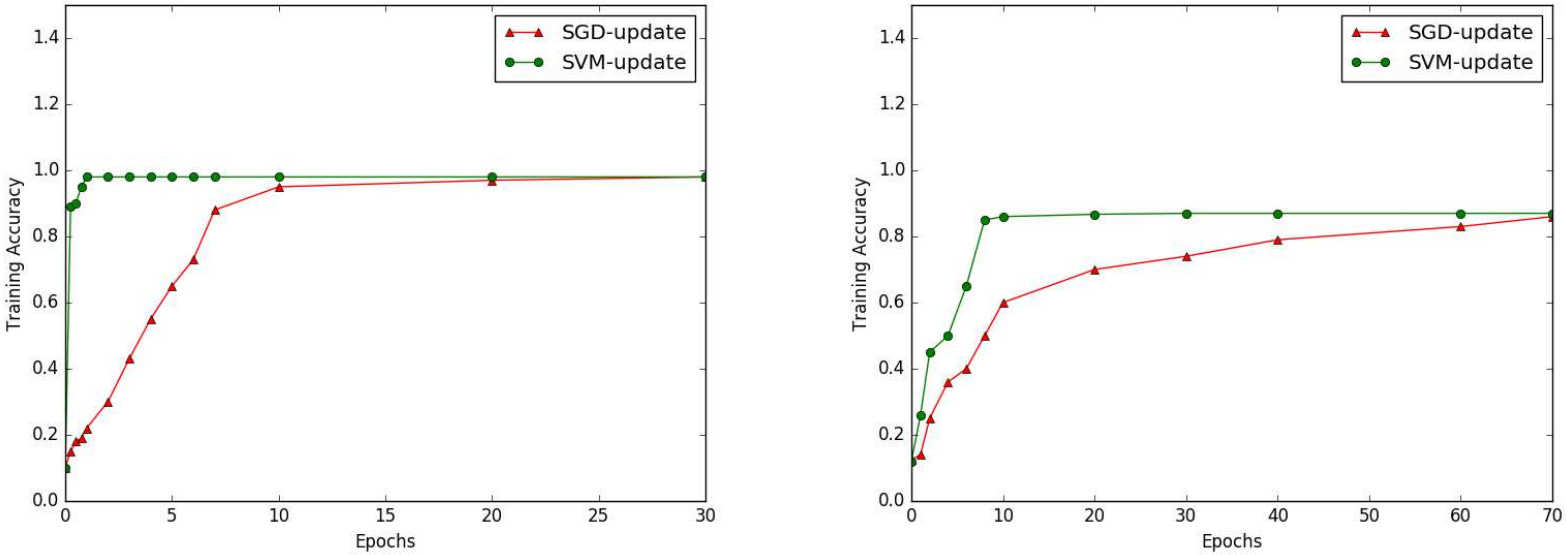


Figure: **Left**: MNIST & **Right**: CIFAR. We plot the training accuracy over the entire train set versus epochs.

Generalization

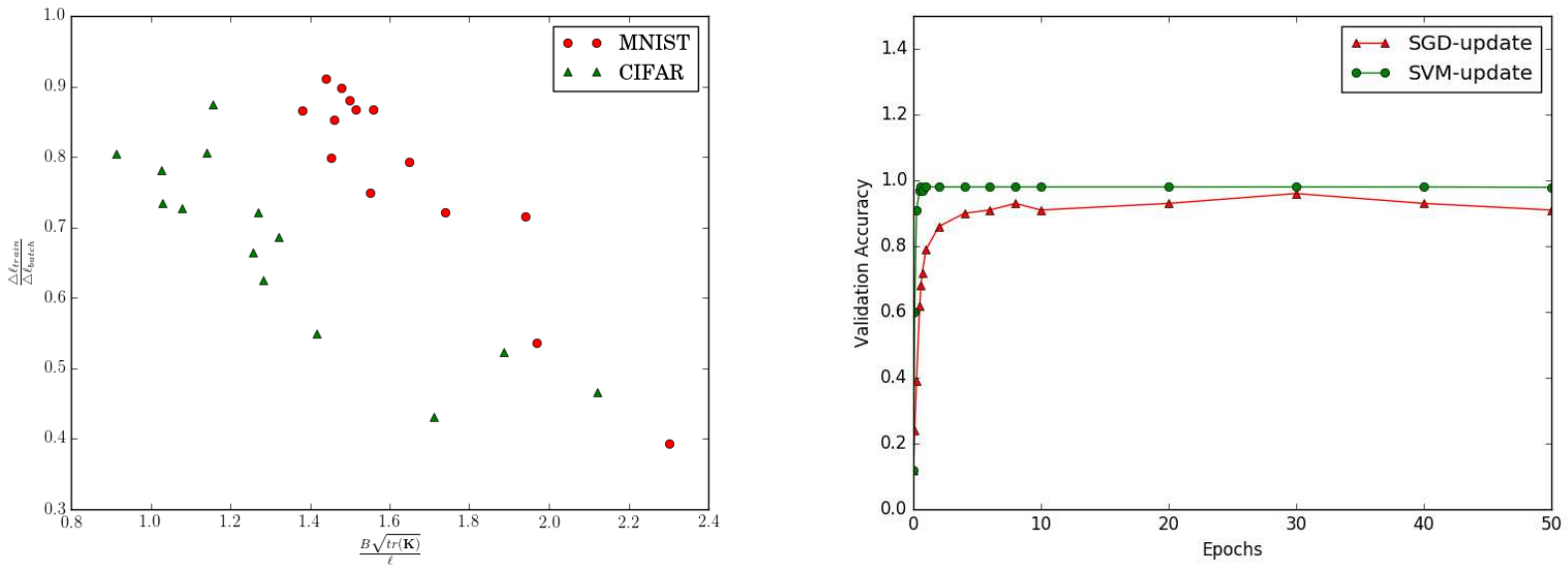


Figure: **Left**: We see that the ratio of decrease in loss over train set and mini-batch decreases with increase in the bound, implying that the updates become less generalized. **Right**: We observe that the validation accuracy initially increases and then stabilizes for mnist using our algorithm, as opposed to sgd.

Robustness: Adversarial Noise

Norm	MNIST		CIFAR	
	sgd-based	svm-based	sgd-based	svm-based
frobenius	5.12	7.32	7.95	9.25
infinity	5.90	8.11	6.50	8.57
nuclear	7.29	8.11	8.50	9.12
1-norm	6.23	7.59	7.40	9.10

Table: We give details of the additive adversarial noise learned for **left**: MNIST and **right**: CIFAR using traditional back-propagation and svm-based updates. Additive adversarial noise is the minimum amount of noise to be added to images such that the network misclassifies them.

Summary and Conclusions

- Introduction to pattern analysis and machine learning through the perspective of kernel methods
- Emphasis on the need to assess and improve generalisation: analysis provided through Radmacher Complexity
- Attempts to use principled methods for complex tasks such as Reinforcement Learning have met with considerable success
- Deep learning follows similar principles of fitting data, but appears to exhibit a remarkable resistance to overfitting when trained using stochastic gradient descent and variants thereof.

Summary and Conclusions

- Introduction to pattern analysis and machine learning through the perspective of kernel methods
- Emphasis on the need to assess and improve generalisation: analysis provided through Radmacher Complexity
- Attempts to use principled methods for complex tasks such as Reinforcement Learning have met with considerable success
- Deep learning follows similar principles of fitting data, but appears to exhibit a remarkable resistance to overfitting when trained using stochastic gradient descent and variants thereof.

Summary and Conclusions

- Introduction to pattern analysis and machine learning through the perspective of kernel methods
- Emphasis on the need to assess and improve generalisation: analysis provided through Radmacher Complexity
- Attempts to use principled methods for complex tasks such as Reinforcement Learning have met with considerable success
- Deep learning follows similar principles of fitting data, but appears to exhibit a remarkable resistance to overfitting when trained using stochastic gradient descent and variants thereof.

Summary and Conclusions

- Introduction to pattern analysis and machine learning through the perspective of kernel methods
- Emphasis on the need to assess and improve generalisation: analysis provided through Radmacher Complexity
- Attempts to use principled methods for complex tasks such as Reinforcement Learning have met with considerable success
- Deep learning follows similar principles of fitting data, but appears to exhibit a remarkable resistance to overfitting when trained using stochastic gradient descent and variants thereof.