Flexible discriminative learning with structured output support vector machines

A short introduction and tutorial

Andrea Vedaldi
University of Oxford

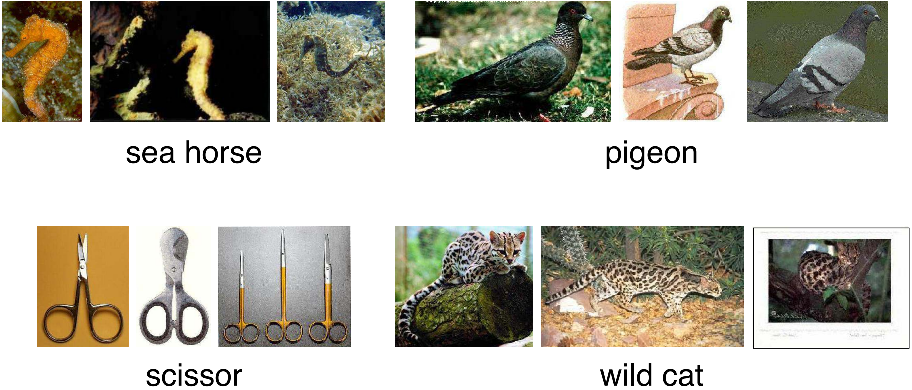
Slides and code at
`http://www.vlfeat.org/~vedaldi/teach.html`

January 2013

Abstract

This tutorial introduces Structured Support Vector Machines (SSVMs) as a tool to effectively learn functions over arbitrary spaces. For example, one can use a SSVM to rank a set of items by decreasing relevance, to localise an object such as a cat in an image, or to estimate the pose of a human in a video. The tutorial reviews the standard notion of SVM and shows how this can be extended to arbitrary output spaces, introducing the corresponding learning formulations. It then gives a complete example on how to design and learn a SSVM with off-the-shelf solvers in MATLAB. The last part discusses how such solvers can be implemented, focusing in particular on the cutting plane and BMRM algorithms.

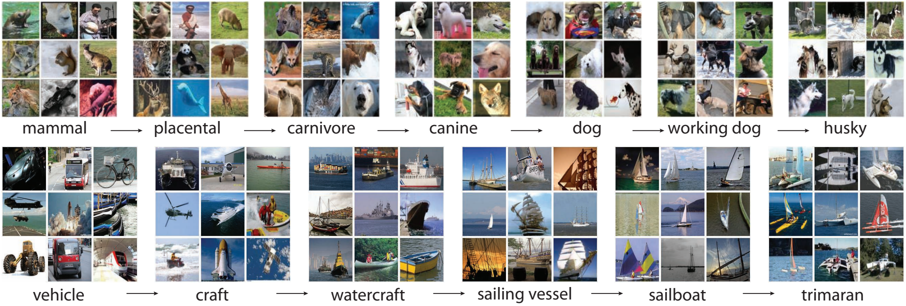
Classification



Caltech-101 (101 classes, 3k images)

E.g. [Weber et al., 2000, Csurka et al., 2004, Fei-Fei et al., 2004, Sivic and Zisserman, 2004]

Classification on a large scale



ImageNet (80k classes, 3.2M images)

E.g. [Deng et al., 2009, Sánchez and Perronnin, 2011, Mensink et al., 2012]

Classification with an SVM

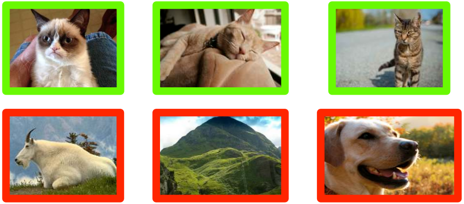
is there a cat?

 **x**

Classification with an SVM

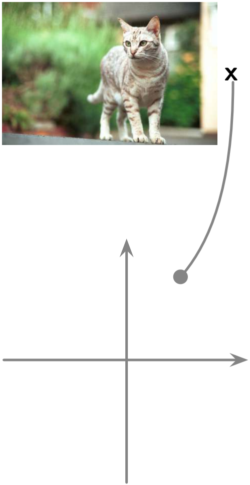
is there a cat?

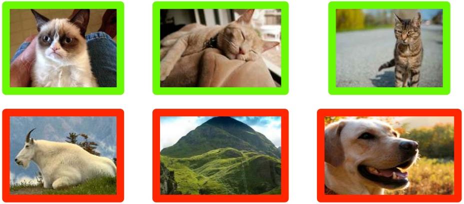
x



Classification with an SVM

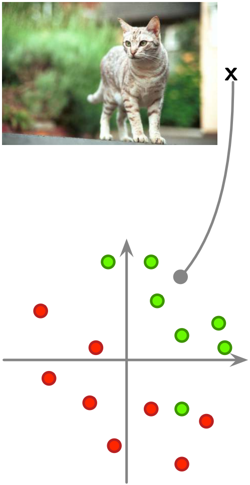
is there a cat?

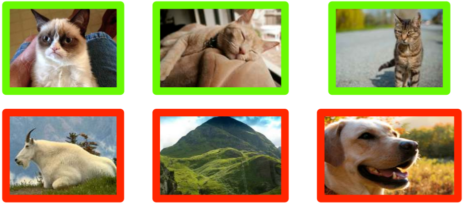




Classification with an SVM

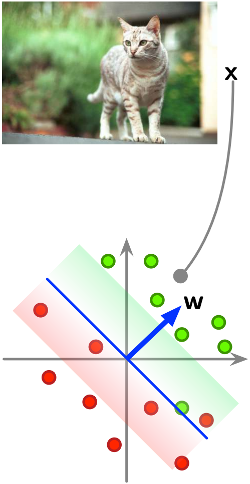
is there a cat?

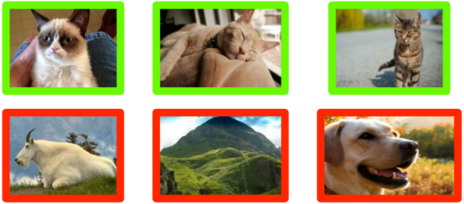




Classification with an SVM

is there a cat?



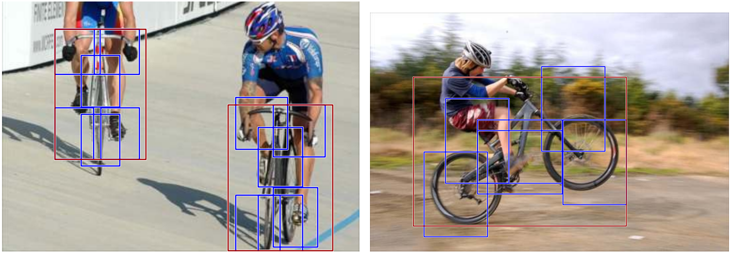


$$F(\mathbf{x}; \mathbf{w}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

Support vector machines can do classification.

What about ...

Object category detection



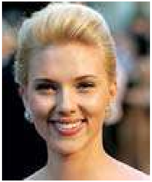
Find objects of a given type (e.g. bicycle) in an image.

E.g. [Leibe and Schiele, 2004, Felzenszwalb et al., 2008, Vedaldi et al., 2009]

Pose estimation

E.g. [Ramanan et al., 2005, Ramanan, 2006, Ferrari et al., 2008]

Relative attributes



(a) Smiling

(b) **?**

(c) Not smiling

(d) Natural

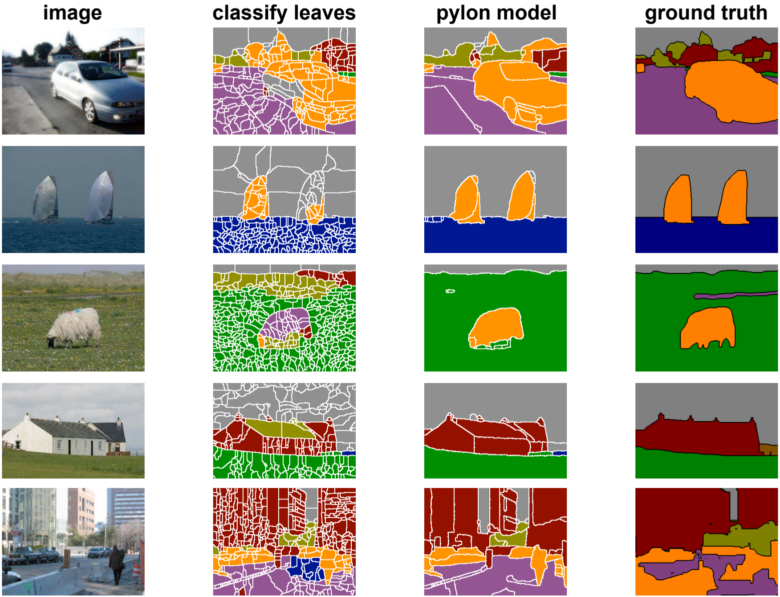
(e) **?**

(f) Manmade

Sometimes it is less ambiguous to **rank** rather than to **classify** objects.

[Parikh and Grauman, 2011]

Segmentation



E.g. [Taskar et al., 2003] (image [Lempitsky et al., 2011])

Learning to handle complex data

- Algorithms that can "understand" images, videos, etc. are too complex to be designed entirely manually.
- **Machine learning** (ML) automatises part of the design based on empirical evidence:

$$\boxed{\text{algorithmic class}} + \boxed{\text{example data}} + \boxed{\text{optimisation}} \xrightarrow{\text{learning}} \text{algorithm.}$$

Support Vector Machines

- There are countless ML methods:
  - Nearest neighbors, perceptron, bagging, boosting, AdaBoost, logistic regression, Support Vector Machines (SVMs), random forests, metric learning, ...
  - Markov random fields, Bayesian networks, Gaussian Processes, ...
  - E.g. [Schölkopf and Smola, 2002b, Hastie et al., 2001]

We will focus on **SVMs and their generalisations**.

1. Good accuracy (when applicable).
2. Clean formulation.
3. Large scale.

Structured output SVMs

Extending SVMs to handle arbitrary output spaces, particularly ones with non-trivial structure (e.g. space of poses, textual translations, sentences in a grammar, etc.).

Outline

Support vector classification

Beyond classification: structured output SVMs

Learning formulations

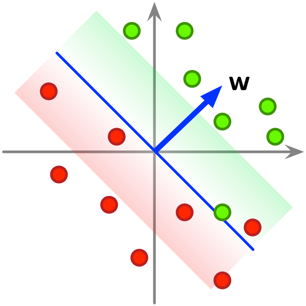
Optimisation

A complete example

Further insights on optimisation

Outline

Scoring function and classification



- The **input** $\mathbf{x} \in \mathbb{R}^d$ is a vector to be classified.
- The **parameter** $\mathbf{w} \in \mathbb{R}^d$ is a vector.
- The **score** is $\langle \mathbf{x}, \mathbf{w} \rangle$.
- The **output** $\hat{y}(\mathbf{x}; \mathbf{w})$ is either
  $+1$ (relevant) or
  -1 (not relevant).

The "machine" part of an SVM is a simple **classification rule** that test the sign of the score:

$$\hat{y}(\mathbf{x}; \mathbf{w}) = \operatorname{sign}\langle \mathbf{x}, \mathbf{w} \rangle$$

E.g. [Schölkopf and Smola, 2002a].

Feature maps

$$\mathbf{x} \xmapsto{\text{feature map}} \Phi(\mathbf{x}) \in \mathbb{R}^d$$

- In the SVM $\langle \mathbf{x}, \mathbf{w} \rangle$ the **input** $\mathbf{x}$ is a *vectorial representation* of a datum.
- Alternatively, one can introduce a feature map:

$$\Phi : \mathcal{X} \to \mathbb{R}^d, \quad \mathbf{x} \mapsto \Phi(\mathbf{x}).$$

The classification rule becomes

$$\hat{y}(\mathbf{x}; \mathbf{w}) = \operatorname{sign}\langle \Phi(\mathbf{x}), \mathbf{w} \rangle.$$

With a feature map, the nature of the input $\mathbf{x} \in \mathcal{X}$ is irrelevant (image, video, audio, ...).

Learning formulation

- The other defining aspect of an SVM is the **objective function** used to learn it.
- Given **example pairs** $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, the objective function is

$$\boxed{E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}.}$$

- Learning the SVM amounts to minimising $E(\mathbf{w})$ to obtain the optimal parameter $\mathbf{w}^*$.

Learning formulation

- The other defining aspect of an SVM is the **objective function** used to learn it.
- Given **example pairs** $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, the objective function is

$$\boxed{E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}.}$$

- Learning the SVM amounts to minimising $E(\mathbf{w})$ to obtain the optimal parameter $\mathbf{w}^*$.

An aside: support vectors

One can show that the minimiser has a sparse decomposition

$$\mathbf{w}^* = \beta_1\mathbf{x}_1 + \cdots + \beta_n\mathbf{x}_n$$

where only a few of the $\beta_i \neq 0$. The corresponding $\mathbf{x}_i$ are the support vectors.

Hinge loss

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \overbrace{\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}}^{\text{average loss}}$$

Intuition

When the hinge loss is small, then the scoring function fits the example data well, with a “safety margin”.

Hinge loss

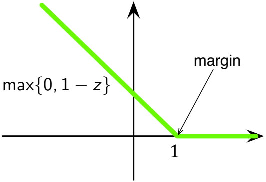
$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \overbrace{\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}}^{\text{average loss}}$$

Intuition

When the hinge loss is small, then the scoring function fits the example data well, with a “safety margin”.

Hinge loss

$$L_i(\mathbf{w}) = \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}.$$



Hinge loss

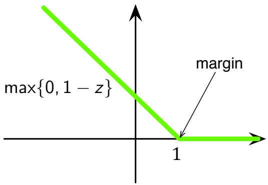
$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \overbrace{\frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i\langle\mathbf{x}_i, \mathbf{w}\rangle\}}^{\text{average loss}}$$

Intuition

When the hinge loss is small, then the scoring function fits the example data well, with a "safety margin".

Hinge loss

$$L_i(\mathbf{w}) = \max\{0, 1 - y_i\langle\mathbf{x}_i, \mathbf{w}\rangle\}.$$



Margin condition

$$\begin{aligned} L_i(\mathbf{w}) = 0 \quad &\Rightarrow \quad \overbrace{y_i\langle\mathbf{x}_i, \mathbf{w}\rangle \geq 1}^{\text{margin condition}} \\ &\Rightarrow \quad \operatorname{sign}\langle\mathbf{x}_i, \mathbf{w}\rangle = y_i. \end{aligned}$$

Hinge loss

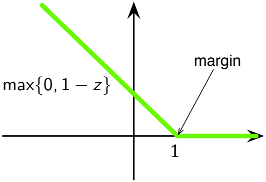
$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \overbrace{\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}}^{\text{average loss}}$$

Intuition

When the hinge loss is small, then the scoring function fits the example data well, with a "safety margin".

Hinge loss

$$L_i(\mathbf{w}) = \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}.$$



Margin condition

$$\begin{aligned} L_i(\mathbf{w}) = 0 \quad &\Rightarrow \quad \overbrace{y_i\langle \mathbf{x}_i, \mathbf{w}\rangle \geq 1}^{\text{margin condition}} \\ &\Rightarrow \quad \operatorname{sign}\langle \mathbf{x}_i, \mathbf{w}\rangle = y_i. \end{aligned}$$

Convexity

The hinge loss is a convex function!

The regulariser

$$E(\mathbf{w}) = \overbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}^{\text{regulariser}} + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}$$

Intuition

If the **regulariser** $\|\mathbf{w}\|^2$ is small, then the scoring function $\langle \mathbf{w}, \mathbf{x}\rangle$ varies slowly.

The regulariser

$$E(\mathbf{w}) = \overbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}^{\text{regulariser}} + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}$$

Intuition

If the **regulariser** $\|\mathbf{w}\|^2$ is small, then the scoring function $\langle \mathbf{w}, \mathbf{x}\rangle$ varies slowly.

To see this:

1. The regulariser is the norm of the derivative of the scoring function:

$$\|\nabla_{\mathbf{x}}\langle \mathbf{x}, \mathbf{w}\rangle\|^2 = \|\mathbf{w}\|^2.$$

The regulariser

$$E(\mathbf{w}) = \overbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}^{\text{regulariser}} + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}$$

Intuition

If the **regulariser** $\|\mathbf{w}\|^2$ is small, then the scoring function $\langle \mathbf{w}, \mathbf{x}\rangle$ varies slowly.

To see this:

1. The regulariser is the norm of the derivative of the scoring function:

$$\|\nabla_{\mathbf{x}}\langle \mathbf{x}, \mathbf{w}\rangle\|^2 = \|\mathbf{w}\|^2.$$

2. Using the Cauchy-Schwarz inequality:

$$(\langle \mathbf{x}, \mathbf{w}\rangle - \langle \mathbf{x}', \mathbf{w}\rangle)^2 \leq \|\mathbf{x} - \mathbf{x}'\|^2 \; \|\mathbf{w}\|^2.$$

The feature map

- The feature map encodes a notion of **similarity**:

$$(\langle \Phi(\mathbf{x}), \mathbf{w} \rangle - \langle \Phi(\mathbf{x}'), \mathbf{w} \rangle)^2 \leq \overbrace{\|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|^2}^{\text{similarity of inputs}} \times \overbrace{\|\mathbf{w}\|^2}^{\text{regularizer}} .$$

Intuition

Inputs with similar features receive similar scores.

The feature map

- The feature map encodes a notion of **similarity**:

$$(\langle \Phi(\mathbf{x}), \mathbf{w} \rangle - \langle \Phi(\mathbf{x}'), \mathbf{w} \rangle)^2 \leq \overbrace{\|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|^2}^{\text{similarity of inputs}} \times \overbrace{\|\mathbf{w}\|^2}^{\text{regularizer}} .$$

Intuition

Inputs with similar features receive similar scores.

Note: in all cases, points whose difference $\Phi(\mathbf{x}) - \Phi(\mathbf{x}')$ is orthogonal to $\mathbf{w}$ receive the same score. This is a $d - 1$ dimensional subspace of irrelevant variations!

SVM summary

The goal is to find a scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ that:

Fits the data by a marging

The scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ should fit the data by a margin:

$$\begin{array}{lll} \text{if } \mathbf{y}_i > 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \geq 1 \\ \text{if } \mathbf{y}_i < 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \leq -1 \end{array}$$

SVM summary

The goal is to find a scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ that:

Fits the data by a marging

The scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ should fit the data by a margin:

$$\begin{array}{lll} \text{if } \mathbf{y}_i > 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \geq 1 \\ \text{if } \mathbf{y}_i < 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \leq -1 \end{array}$$

Is regular

A small variation of the feature $\Phi(\mathbf{x})$ should not change the score $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ too much. The regulariser $\|\mathbf{w}\|^2$ is a bound on this variation.

SVM summary

The goal is to find a scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ that:

Fits the data by a marging

The scoring function $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ should fit the data by a margin:

$$\begin{array}{lll} \text{if } \mathbf{y}_i > 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \geq 1 \\ \text{if } \mathbf{y}_i < 0 & \text{then} & \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle \leq -1 \end{array}$$

Is regular

A small variation of the feature $\Phi(\mathbf{x})$ should not change the score $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ too much. The regulariser $\|\mathbf{w}\|^2$ is a bound on this variation.

Reflects prior information

Whether a variation of the input $\mathbf{x}$ is considered to be small or large depends on the choice of the feature map $\Phi(\mathbf{x})$. This establishes *a-priori* which inputs should receive similar scores.

Outline

Beyond classification

Consider now the general problem of learning a function

$$f: \qquad \mathcal{X} \to \mathcal{Y}, \qquad \mathbf{x} \mapsto \mathbf{y},$$

where both the input and output spaces are general. Examples:

Beyond classification

Consider now the general problem of learning a function

$$f: \qquad \mathcal{X} \to \mathcal{Y}, \qquad \mathbf{x} \mapsto \mathbf{y},$$

where both the input and output spaces are general. Examples:

Ranking.

- given a set of objects $(o_1, \ldots, o_k)$ as **input** $\mathbf{x}$,
- return a order as **output** $\mathbf{y}$.

Beyond classification

Consider now the general problem of learning a function

$$f: \quad \mathcal{X} \to \mathcal{Y}, \quad \mathbf{x} \mapsto \mathbf{y},$$

where both the input and output spaces are general. Examples:

Ranking.

- given a set of objects $(o_1, \ldots, o_k)$ as **input** $\mathbf{x}$,
- return a order as **output** $\mathbf{y}$.

Pose estimation.

- given an image of a human as **input** $\mathbf{x}$,
- return the parameters $(p_1, \ldots, p_k)$ of his/her pose as **output** $\mathbf{y}$.

Beyond classification

Consider now the general problem of learning a function

$$f: \qquad \mathcal{X} \to \mathcal{Y}, \qquad \mathbf{x} \mapsto \mathbf{y},$$

where both the input and output spaces are general. Examples:

Ranking.

- given a set of objects $(o_1, \ldots, o_k)$ as **input x**,
- return a order as **output y**.

Pose estimation.

- given an image of a human as **input x**,
- return the parameters $(p_1, \ldots, p_k)$ of his/her pose as **output y**.

Image segmentation.

- given an image from Flikr as **input x**,
- return a mask highlighting the "foreground object" as **output y**.

Support Vector Regression /1

A real function $\mathbb{R}^d \to \mathbb{R}$ can be approximated *directly* by the SVM score:

$$f(\mathbf{x}) \approx \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle.$$

- Think of the feature map $\Phi(\mathbf{x})$ as a collection of *basis functions*. For instance, if $x \in \mathbb{R}$, one can use the basis of second order polynomials:

$$\Phi(x) = \begin{bmatrix} 1 & x & x^2 \end{bmatrix}^\top \quad \Rightarrow \quad \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle = w_1 + w_2 x + 2_3 x^2.$$

Support Vector Regression /1

A real function $\mathbb{R}^d \to \mathbb{R}$ can be approximated *directly* by the SVM score:

$$f(\mathbf{x}) \approx \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle.$$

- Think of the feature map $\Phi(\mathbf{x})$ as a collection of *basis functions*. For instance, if $x \in \mathbb{R}$, one can use the basis of second order polynomials:

$$\Phi(x) = \begin{bmatrix} 1 & x & x^2 \end{bmatrix}^\top \quad \Rightarrow \quad \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle = w_1 + w_2 x + 2_3 x^2.$$

- The goal is to find $\mathbf{w}$ (e.g. polynomial coefficients) such that the score fits the example data

$$\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle \approx y_i$$

  by minimising the L^1 error

$$L_i(\mathbf{w}) = |y_i - \langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle|.$$

Support Vector Regression /2

SVR is just a variant of regularised regressions:

method	**loss**	**regul.**	**objective function**
SVR	l^1	l^2	$\frac{1}{n}\sum_{i=1}^{n} \lvert y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i) \rvert + \frac{\lambda}{2}\lVert \mathbf{w} \rVert_2^2$
least square	l^2	none	$\frac{1}{2n}\sum_{i=1}^{n} (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2$
ridge regression	l^2	l^2	$\frac{1}{2n}\sum_{i=1}^{n} (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2 + \frac{\lambda}{2}\lVert \mathbf{w} \rVert_2^2$
lassoo	l^2	l^1	$\frac{1}{2n}\sum_{i=1}^{n} (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2 + \lambda \lVert \mathbf{w} \rVert_1$

Limitation: only real functions!

Support Vector Regression /2

SVR is just a variant of regularised regressions:

method	loss	regul.	objective function
SVR	l^1	l^2	$\frac{1}{n}\sum_{i=1}^n \lvert y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i)\rvert + \frac{\lambda}{2}\lVert\mathbf{w}\rVert_2^2$
least square	l^2	none	$\frac{1}{2n}\sum_{i=1}^n (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2$
ridge regression	l^2	l^2	$\frac{1}{2n}\sum_{i=1}^n (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2 + \frac{\lambda}{2}\lVert\mathbf{w}\rVert_2^2$
lassoo	l^2	l^1	$\frac{1}{2n}\sum_{i=1}^n (y_i - \mathbf{w}^\top \Phi(\mathbf{x}_i))^2 + \lambda\lVert\mathbf{w}\rVert_1$

Limitation: only real functions!

An aside: ϵ-insensitive L^1 loss

Actually, SVR makes use of a slightly more general loss

$$L_i(\mathbf{w}) = \max\{0, \lvert y_i - \langle \mathbf{w}, \mathbf{x}_i\rangle\rvert - \epsilon\}$$

which is insensitive to error below a threshold ϵ. One can set $\epsilon = 0$ though [Smola and Scholkopf, 2004].

A general approach: learning the graph

Use a binary SVM to classify which pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ belongs to the *graph* of the function (treat the output as an input!):

$$\mathbf{y} = f(\mathbf{x}) \quad \Leftrightarrow \quad \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle > 0.$$

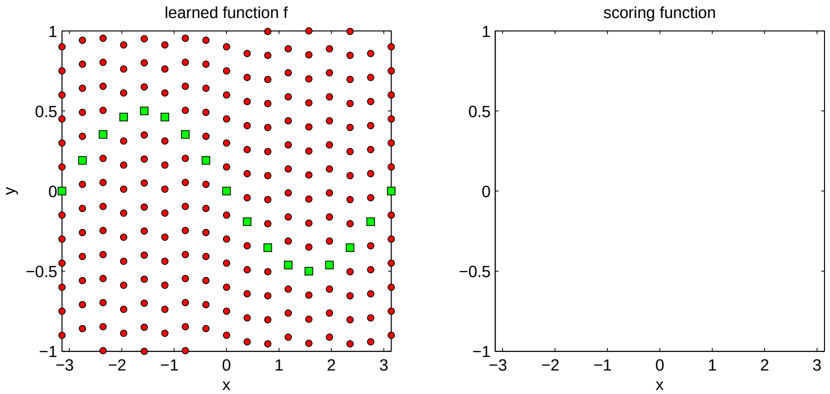
Joint feature map

In order to classify pairs $(\mathbf{x}, \mathbf{y})$, these must be encoded as vectors. To this end, we need a **joint feature map**:

$$\Phi : (\mathbf{x}, \mathbf{y}) \rightarrow \Phi(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d$$

As long as this feature can be designed, the nature of $\mathbf{x}$ and $\mathbf{y}$ is irrelevant.

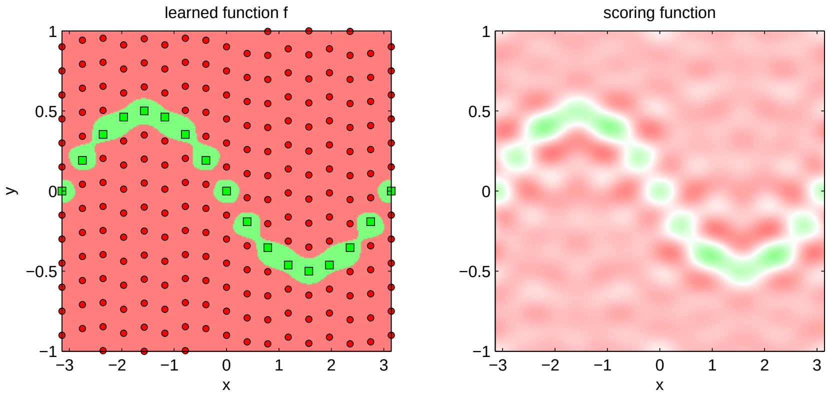
Example: learning the graph of a real function /1



Algorithm:

1. Start from the true pairs (x_i, y_i) (green squares) where the graph should pass.
2. Add many false pairs (x_i, y_i) (red dots) where the graph should *not* pass.

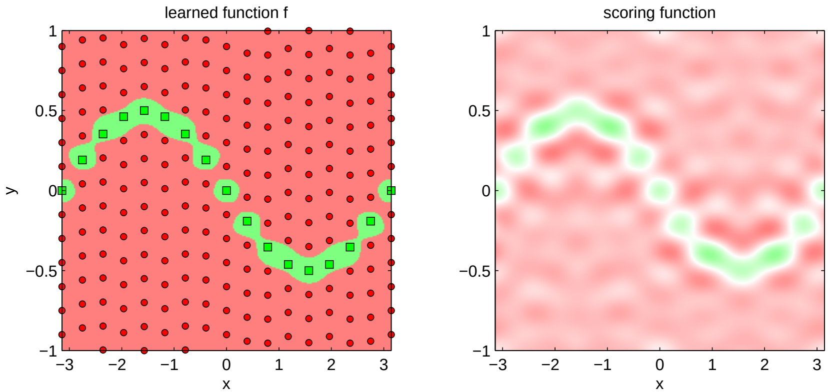
Example: learning the graph of a real function /1



Algorithm:

1. Start from the true pairs (x_i, y_i) (green squares) where the graph should pass.
2. Add many false pairs (x_i, y_i) (red dots) where the graph should *not* pass.
3. Learn a scoring function $\langle \mathbf{w}, \Psi(x, y) \rangle$ to fit these points.
4. Define the learned function graph to be the collection of points such that $\langle \mathbf{w}, \Psi(x, y) \rangle > 0$ (green areas).

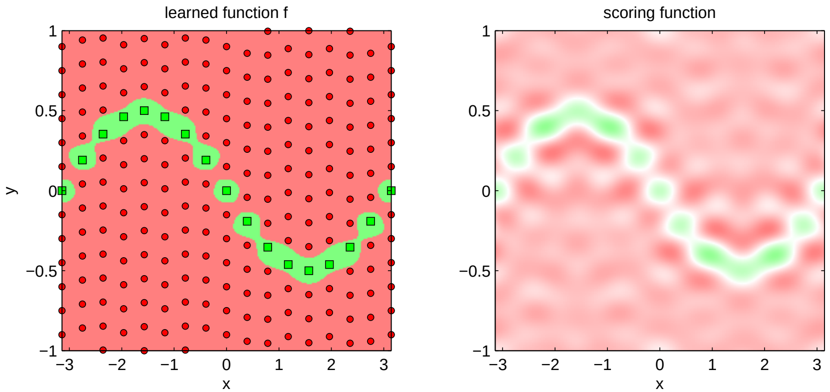
Example: learning the graph of a real function /2



In this example the joint feature map is a Fourier basis (note the ringing!)

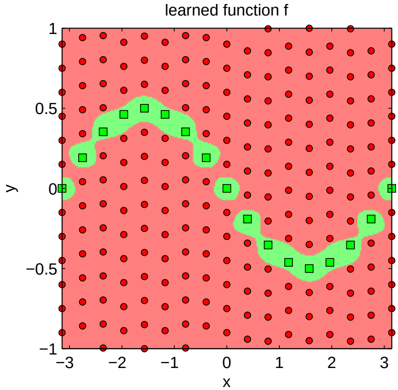
$$\Psi(x, y) = \begin{bmatrix} \cos(f_{1x}x + f_{1y}y + \phi_1) \\ \cos(f_{2x}x + f_{2y}y + \phi_2) \\ \vdots \\ \cos(f_{dx}x + f_{dy}y + \phi_d) \end{bmatrix}, \quad \text{for appropriate } (f_{1i}, f_{2i}, \phi_i).$$

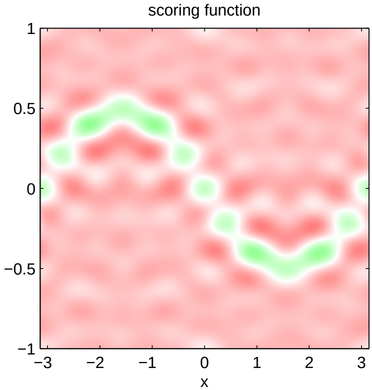
The good and the bad



The good: **works for any type of inputs and outputs!** (Not just real functions.)

The good and the bad





The good: **works for any type of inputs and outputs!** (Not just real functions.)

The Bad:

- **Not one-to-one.** For each **x**, there are *multiple outputs* **y** with positive score.
- **Not complete.** There are **x** for which all the outputs have negative score.
- **Very large negative example set.**

Structured output SVMs

Structured output SVM. Issues 1 and 2 can be fixed by choosing the highest scoring output for each input:

$$\boxed{\hat{\mathbf{y}}(\mathbf{x};\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle}$$

Structured output SVMs

Structured output SVM. Issues 1 and 2 can be fixed by choosing the highest scoring output for each input:

$$\boxed{\hat{\mathbf{y}}(\mathbf{x};\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle}$$

Intuition

The **scoring function**

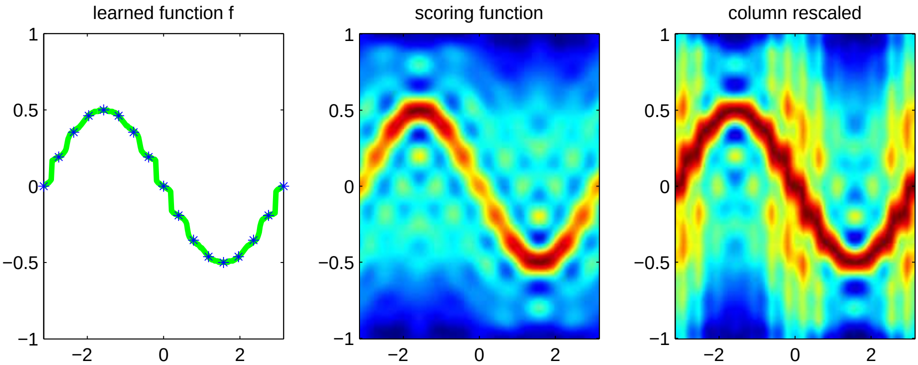
$$\langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle$$

is somewhat analogous to a posterior probability density function

$$P(\mathbf{y}|\mathbf{x})$$

but it does <u>not</u> have any probabilistic meaning.

Example: real function



- $f(x) = y$ that maximises the score along column x.
- $f(x)$ is now uniquely and completely defined.
- **Note:** only the relative values of the score along a column really matter (see rescaled version on the right).

Inference problem

- **Inference problem.** Evaluating a structured SVM requires solving the problem

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle.$$

- The efficiency of using a structured SVM (after learning) depends on how quickly the inference problem can be solved.

Example: binary linear SVM

Standard SVMs can be easily interpreted as a structured SVMs:

- Output space:

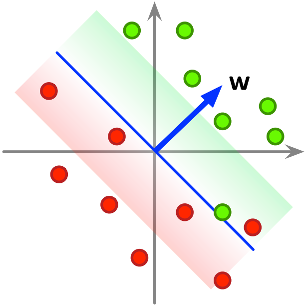
$$y \in \mathcal{Y} = \{-1, +1\}.$$

- Feature map:

$$\Psi(\mathbf{x}, y) = \frac{y}{2}\mathbf{x}.$$

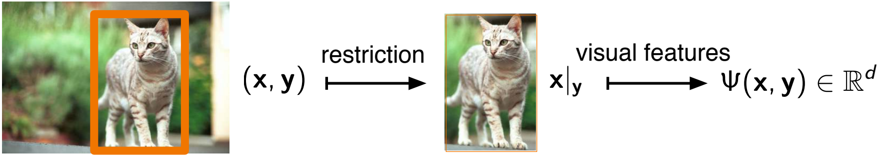
- Inference:

$$\hat{y}(\mathbf{x}; \mathbf{w}) = \underset{y \in \{-1,+1\}}{\operatorname{argmax}} \frac{y}{2}\langle \mathbf{w}, \mathbf{x} \rangle = \operatorname{sign}\langle \mathbf{w}, \mathbf{x} \rangle.$$



Example: object localisation

Let $\mathbf{x}$ be an image and $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^4$ a rectangular window. The goal is to find the window containing a given object.



- Let $\mathbf{x}|_{\mathbf{y}}$ denote an image window (crop).

Example: object localisation

Let $\mathbf{x}$ be an image and $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^4$ a rectangular window. The goal is to find the window containing a given object.

$(\mathbf{x}, \mathbf{y}) \xmapsto{\text{restriction}}$

$\mathbf{x}|_{\mathbf{y}} \xmapsto{\text{visual features}} \Psi(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d$

- Let $\mathbf{x}|_{\mathbf{y}}$ denote an image window (crop).
- Standard SVM: score one window:

$$\begin{aligned} \Phi(\mathbf{x}|_{\mathbf{y}}) &= \text{“histogram of SIFT features”}, \\ \langle \mathbf{w}, \Phi(\mathbf{x}|_{\mathbf{y}}) \rangle &= \text{“window score”}. \end{aligned}$$

Example: object localisation

Let $\mathbf{x}$ be an image and $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^4$ a rectangular window. The goal is to find the window containing a given object.

$$(\mathbf{x}, \mathbf{y}) \xmapsto{\text{restriction}}$$

$$\mathbf{x}|_{\mathbf{y}} \xmapsto{\text{visual features}} \Psi(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d$$

- Let $\mathbf{x}|_{\mathbf{y}}$ denote an image window (crop).
- Standard SVM: score one window:

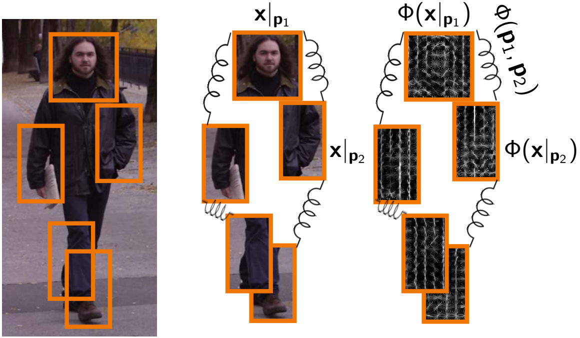
$$\Phi(\mathbf{x}|_{\mathbf{y}}) = \text{“histogram of SIFT features”},$$
$$\langle \mathbf{w}, \Phi(\mathbf{x}|_{\mathbf{y}}) \rangle = \text{“window score”}.$$

- Structured SVM: try all windows and pick the best one:

$$\hat{\mathbf{y}}(\mathbf{x}; \mathbf{w}) = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \langle \mathbf{w}, \Phi(\mathbf{x}|_{\mathbf{y}}) \rangle.$$

Example: pose estimation

Let $\mathbf{x}$ be an image and $\mathbf{y} = (\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5)$ the pose of a human, expressed as the 2D location of five parts.



$$\Psi(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \Phi(\mathbf{x}|_{\mathbf{p}_1}) \\ \Phi(\mathbf{x}|_{\mathbf{p}_1}) \\ \vdots \\ \Phi(\mathbf{x}|_{\mathbf{p}_5}) \\ \Phi(\mathbf{p}_1, \mathbf{p}_2) \\ \vdots \\ \Phi(\mathbf{p}_5, \mathbf{p}_6) \end{bmatrix}$$

Inituition

The score $\langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle$ reflects how well the five image parts match their appearance models and whether the deformation is reasonable or not.

Example: ranking /1

- Consider the problem of ranking a list of objects $\mathbf{x} = (o_1, \ldots, o_n)$ (input).
- The output $\mathbf{y}$ is an ranking (total order). This can be represented as a matrix $\mathbf{y}$ such that

$$
\begin{aligned}
y_{ij} &= +1, \qquad o_i \text{ has higher rank than } o_j, \\
y_{ij} &= -1, \qquad \text{otherwise.}
\end{aligned}
$$

Example: ranking /1

- Consider the problem of ranking a list of objects $\mathbf{x} = (o_1, \ldots, o_n)$ (input).
- The output $\mathbf{y}$ is an ranking (total order). This can be represented as a matrix $\mathbf{y}$ such that

$$
\begin{aligned}
y_{ij} &= +1, && o_i \text{ has higher rank than } o_j, \\
y_{ij} &= -1, && \text{otherwise.}
\end{aligned}
$$

A joint feature map for ranking

$$\Psi(\mathbf{x}, \mathbf{y}) = \sum_{ij} y_{ij} \langle \Phi(o_i) - \Phi(o_j), \mathbf{w} \rangle.$$

Example: ranking /2

This structured SVM ranks the objects by decreasing score $\langle \Phi(o_i), \mathbf{w} \rangle$:

$$\hat{y}_{ij}(\mathbf{x}; \mathbf{w}) = \text{sign}\left(\langle \Phi(o_i), \mathbf{w} \rangle - \langle \Phi(o_j), \mathbf{w} \rangle \right).$$

In fact the score of this output

$$\begin{aligned} \langle \mathbf{w}, \Psi(\mathbf{x}, \hat{\mathbf{y}}(\mathbf{x}; \mathbf{w})) \rangle &= \sum_{ij} y_{ij} \langle \Phi(o_i) - \Phi(o_j), \mathbf{w} \rangle \\ &= \sum_{ij} \text{sign}\langle \Phi(o_i) - \Phi(o_j), \mathbf{w} \rangle \langle \Phi(o_i) - \Phi(o_j), \mathbf{w} \rangle \\ &= \sum_{ij} |\langle \Phi(o_i) - \Phi(o_j), \mathbf{w} \rangle| \end{aligned}$$

is maximum.

Outline

Summary so far and what remains to be done

Input-output relation

The SVM defines an input-output relation based on maximising the joint score:

$$\hat{\mathbf{y}}(\mathbf{x}; \mathbf{w}) = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle.$$

Next: how to fit the input-output relation to data.

Learning formulation /1

Given n example input-output pairs

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n),$$

find $\mathbf{w}$ such that the structured SVM approximately fit them

$$\hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}) \approx \mathbf{y}_i, \quad i = 1, \ldots, n,$$

while controlling the complexity of the estimated function.

Learning formulation /1

Given n example input-output pairs

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n),$$

find $\mathbf{w}$ such that the structured SVM approximately fit them

$$\hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}) \approx \mathbf{y}_i, \quad i = 1, \ldots, n,$$

while controlling the complexity of the estimated function.

Objective function (non-convex)

$$E_1(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}))$$

Notation reminder: Δ is the loss function, $\hat{\mathbf{y}}$ the output estimated by the SVM, $\mathbf{y}_i$ the ground truth output, and $\mathbf{x}_i$ the ground truth input.

Loss function

The **loss function** measures the fit quality:

$$\Delta(\mathbf{y}, \hat{\mathbf{y}})$$

such that $\Delta(\mathbf{y}, \hat{\mathbf{y}}) \geq 0$ and $\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 0$ if, and only if, $\mathbf{y} = \hat{\mathbf{y}}$.

Examples:

- For a binary SVM the loss is

$$\Delta(y, \hat{y}) = \begin{cases} 1, & y \neq \hat{y}, \\ 0, & \text{otherwise.} \end{cases}$$

Loss function

The **loss function** measures the fit quality:

$$\Delta(\mathbf{y}, \hat{\mathbf{y}})$$

such that $\Delta(\mathbf{y}, \hat{\mathbf{y}}) \geq 0$ and $\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 0$ if, and only if, $\mathbf{y} = \hat{\mathbf{y}}$.

Examples:

- For a binary SVM the loss is

$$\Delta(y, \hat{y}) = \begin{cases} 1, & y \neq \hat{y}, \\ 0, & \text{otherwise.} \end{cases}$$

- In object localisation the loss could be one minus the ratio of the areas of the intersection and union of the rectangles $\mathbf{y}$ and $\hat{\mathbf{y}}$:

$$\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{|\mathbf{y} \cap \hat{\mathbf{y}}|}{|\mathbf{y} \cup \hat{\mathbf{y}}|}.$$

Loss function

The **loss function** measures the fit quality:

$$\Delta(\mathbf{y}, \hat{\mathbf{y}})$$

such that $\Delta(\mathbf{y}, \hat{\mathbf{y}}) \geq 0$ and $\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 0$ if, and only if, $\mathbf{y} = \hat{\mathbf{y}}$.

Examples:

- For a binary SVM the loss is

$$\Delta(y, \hat{y}) = \begin{cases} 1, & y \neq \hat{y}, \\ 0, & \text{otherwise.} \end{cases}$$

- In object localisation the loss could be one minus the ratio of the areas of the intersection and union of the rectangles $\mathbf{y}$ and $\hat{\mathbf{y}}$:

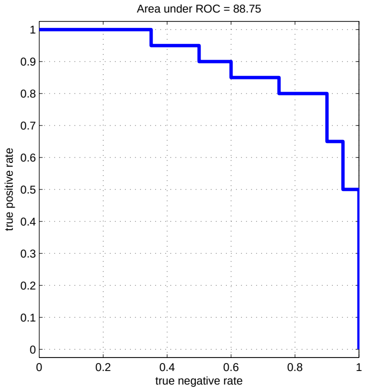
$$\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{|\mathbf{y} \cap \hat{\mathbf{y}}|}{|\mathbf{y} \cup \hat{\mathbf{y}}|}.$$

- In ranking ...

Example: a ranking loss

In ranking, suitable losses include the ROC-AUC, the precision-recall AUC, precision @ k, ...

- The ROC curve plots the true positive rate against the true negative rate.



- Given the “true” ranking $\mathbf{y}$ and the estimated $\hat{\mathbf{y}}$, we can define

$$\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \text{ROCAUC}(\mathbf{y}, \hat{\mathbf{y}})$$

- One can show that this is simply the number of incorrectly ranked pairs, i.e.

$$\Delta(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n^2} \sum_{i,j=1}^{n} [y_{ij} \neq \hat{y}_{ij}]$$

Learning formulation /2

The goal of learning is to find the minimiser $\mathbf{w}^*$ of:

$$E_1(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})),$$

$$\text{where } \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}) = \underset{\mathbf{y}\in\mathcal{Y}}{\operatorname{argmax}} \langle \mathbf{w}, \Phi(\mathbf{x}_i, \mathbf{y})\rangle.$$

The dependency of the loss on $\mathbf{w}$ is very complex: Δ is non-convex and is composed with argmax!

Learning formulation /2

The goal of learning is to find the minimiser $\mathbf{w}^*$ of:

$$E_1(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i;\mathbf{w})),$$

$$\text{where } \hat{\mathbf{y}}(\mathbf{x}_i;\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}}\langle\mathbf{w}, \Phi(\mathbf{x}_i, \mathbf{y})\rangle.$$

The dependency of the loss on $\mathbf{w}$ is very complex: Δ is non-convex and is composed with argmax!

Objective function (convex)

Given a **convex surrogate loss** $L_i(\mathbf{w}) \approx \Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i;\mathbf{w}))$ we consider the objective

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}L_i(\mathbf{w}).$$

The surrogate loss

- The key in the success of the structured SVMs is the existence of good surrogates. There are *standard constructions* that work well in a variety of cases (but not always!).

The surrogate loss

- The key in the success of the structured SVMs is the existence of good surrogates. There are *standard constructions* that work well in a variety of cases (but not always!).
- The aim is to make minimising $L_i(\mathbf{w})$ have the same effect as minimising $\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}))$.
- **Bounding property:**

$$\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})) \leq L_i(\mathbf{w}).$$

The surrogate loss

- The key in the success of the structured SVMs is the existence of good surrogates. There are *standard constructions* that work well in a variety of cases (but not always!).
- The aim is to make minimising $L_i(\mathbf{w})$ have the same effect as minimising $\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w}))$.
- **Bounding property:**

$$\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})) \leq L_i(\mathbf{w}).$$

Tightness

- If we can find $\mathbf{w}^*$ s.t. $L_i(\mathbf{w}^*) = 0$, then $\Delta(\mathbf{y}_i, \mathbf{y}(\mathbf{x}_i; \mathbf{w}^*)) = 0$.
- But can we?
- Not always! Consider setting $L_i(\mathbf{w}) =$ "very large constant".
- We need a tight bound. E.g.:

$$\Delta(\mathbf{y}_i, \mathbf{y}(\mathbf{x}_i; \mathbf{w}^*)) = 0 \qquad \Rightarrow \qquad L_i(\mathbf{w}^*) = 0.$$

Margin rescaling surrogate

- **Margin rescaling** is the first standard surrogate construction:

$$\boxed{L_i(\mathbf{w}) = \sup_{\mathbf{y} \in \mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle.}$$

Margin rescaling surrogate

- **Margin rescaling** is the first standard surrogate construction:

$$\boxed{L_i(\mathbf{w}) = \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.}$$

- This surrogate *bounds* the loss:

$$\begin{aligned}
\Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})) &\le \Delta(\mathbf{y}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})) + \overbrace{\langle \Psi(\mathbf{x}_i, \hat{\mathbf{y}}(\mathbf{x}_i; \mathbf{w})), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle}^{\geq 0 \text{ because } \hat{\mathbf{y}}(\mathbf{x}_i;\mathbf{w}) \text{ maximises the score by definition.}} \\
&\le \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle \\
&= L_i(\mathbf{w})
\end{aligned}$$

Margin condition

- Is margin rescaling a *tight* approximation?
- The following **margin condition** holds

$$L_i(\mathbf{w}^*) = 0 \quad \Leftrightarrow \quad \forall \mathbf{y} \in \mathcal{Y} : \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle}^{\text{score of g.t. output}} \geq \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle}^{\text{score of any other output}} + \overbrace{\Delta(\mathbf{y}_i, \mathbf{y})}^{\text{margin}}$$

Margin condition

- Is margin rescaling a *tight* approximation?
- The following **margin condition** holds

$$L_i(\mathbf{w}^*) = 0 \quad \Leftrightarrow \quad \forall \mathbf{y} \in \mathcal{Y}: \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle}^{\text{score of g.t. output}} \geq \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle}^{\text{score of any other output}} + \overbrace{\Delta(\mathbf{y}_i, \mathbf{y})}^{\text{margin}}$$

Tightness

- The surrogate is <u>not</u> tight in the sense above:

$$\Delta(\mathbf{y}_i, \mathbf{y}(\mathbf{x}_i; \mathbf{w}^*)) = 0 \qquad \not\Rightarrow \qquad L_i(\mathbf{w}^*) = 0.$$

- In order to minimise the surrogate,
<u>the more stringent margin condition has to be satisfied!</u>
- But this is usually good enough, and in fact beneficial (implies robustness).

Slack rescaling surrogate

- **Slack rescaling** is the second standard surrogate construction:

$$\boxed{L_i(\mathbf{w}) = \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) \left[1 + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle\right].}$$

- May give better results than marging rescaling.
- However, it is often significantly harder to treat in calculations.
- The margin condition is

$$L_i(\mathbf{w}^*) = 0 \quad \Leftrightarrow \quad \forall \mathbf{y} \neq \mathbf{y}_i : \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle}^{\text{score of g.t. output}} \geq \overbrace{\langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle}^{\text{score of any other output}} + \overbrace{1}^{\text{margin}}$$

Augmented inference

- Evaluating the objective $E(\mathbf{w})$ requires computing the supremum in the augment loss

$$\sup_{\mathbf{y} \in \mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle.$$

- Maximising this quantity is the **augmented inference** problem due to its similarity with the inference problem

$$\max_{\mathbf{y} \in \mathcal{Y}} \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle$$

- Augmented inference can be significantly harder than inference, especially for slack rescaling.

Example: binary linear SVM

- Recall that for a binary linear SVM:

$$\mathcal{Y} = \{-1, +1\}, \quad \Psi(\mathbf{x}, y) = \frac{y}{2}\mathbf{x}, \quad \Delta(y_i, \hat{y}) = [\mathbf{y}_i \neq y].$$

Example: binary linear SVM

- Recall that for a binary linear SVM:

$$\mathcal{Y} = \{-1, +1\}, \quad \Psi(\mathbf{x}, y) = \frac{y}{2}\mathbf{x}, \quad \Delta(y_i, \hat{y}) = [\mathbf{y}_i \neq y].$$

- Then in the *margin rescaling* construction, solving the augmented inference problem yields

$$\begin{aligned} L_i(\mathbf{w}) &= \sup_{y \in \{-1,1\}} [y_i \neq y] + \frac{y}{2}\langle \mathbf{x}_i \mathbf{w}\rangle - \frac{y_i}{2}\langle \mathbf{x}_i, \mathbf{w}\rangle \\ &= \max_{y \in \{-y_i, y_i\}} [y_i \neq y] + \frac{y - y_i}{2}\langle \mathbf{x}_i, \mathbf{w}\rangle \\ &= \max\{0, 1 - y_i\langle \mathbf{x}_i, \mathbf{w}\rangle\}, \end{aligned}$$

  i.e. the same loss of a standard SVM.
- In this case, slack rescaling yields the same result.

The good and the bad of convex surrogates

Good:

- Convex surrogates *separate the ground truth outputs* $\mathbf{y}_i$ *from other outputs* $\mathbf{y}$ by a margin modulated by the loss.

Bad:

- Despite their construction, they can be poor approximations of the original loss.
- They are unimodal, and therefore cannot model situations in which different outputs are equally acceptable.
- If the ground truth $\mathbf{y}_i$ is not separable, they may be incapable of identifying *which is the best output that can actually be achieved instead* – no graceful fallback.

Outline

Summary so far and what remains to be done

Input-output relation

The SVM defines an input-output relation based on maximising the joint score:

$$\hat{\mathbf{y}}(\mathbf{x};\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y}) \rangle.$$

Convex surrogate objective

The joint score can be designed to fit the data $(\mathbf{x}_1,\mathbf{y}_1),\dots,(\mathbf{x}_n,\mathbf{y}_n)$ by optimising

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w}).$$

Next: how to solve this optimisation problem.

A (naive) direct approach /1

- Learning a structured SVM requires solving an optimisation problem of the type:

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w}), \\
L_i(\mathbf{w}) &= \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.
\end{aligned}
$$

A (naive) direct approach /1

- Learning a structured SVM requires solving an optimisation problem of the type:

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w}),$$
$$L_i(\mathbf{w}) = \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.$$

- More in general, this can be rewritten as

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w}), \qquad L_i(\mathbf{w}) = \sup_{\mathbf{y}\in\mathcal{Y}} b_{i\mathbf{y}} - \langle \mathbf{a}_{i\mathbf{y}}, \mathbf{w}\rangle.$$

A (naive) direct approach /2

This problem can be rewritten as a **constrained quadratic program** in the parameters $\mathbf{w}$ and the slack variables $\boldsymbol{\xi}$:

$$E(\mathbf{w}, \boldsymbol{\xi}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i,$$

$$\xi_i \geq b_{i\mathbf{y}} - \langle \mathbf{a}_{i\mathbf{y}}, \mathbf{w} \rangle \quad \forall i = 1, \ldots, n, \ \mathbf{y} \in \mathcal{Y}.$$

Can we use a standard quadratic solver (e.g. `quadprog` in MATLAB)?

A (naive) direct approach /2

This problem can be rewritten as a **constrained quadratic program** in the parameters $\mathbf{w}$ and the slack variables $\boldsymbol{\xi}$:

$$E(\mathbf{w}, \boldsymbol{\xi}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \xi_i,$$

$$\xi_i \geq b_{i\mathbf{y}} - \langle \mathbf{a}_{i\mathbf{y}}, \mathbf{w} \rangle \quad \forall i = 1, \ldots, n, \ \mathbf{y} \in \mathcal{Y}.$$

Can we use a standard quadratic solver (e.g. `quadprog` in MATLAB)?

The size of this problem

- There is one *set* of constraints for each data point $(\mathbf{x}_i, \mathbf{y}_i)$.
- Each set of constraints contains one linear constraint for each output $\mathbf{y}$.
- Way too large (even infinite!) to be directly fed to a quadratic solver.

A second look

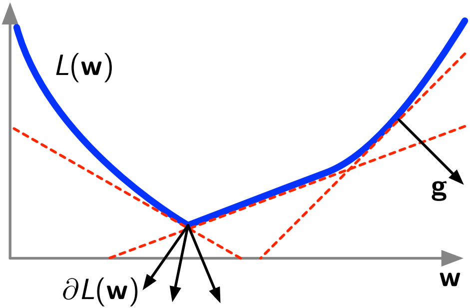
- Let's look again to the original problem is a slightly different form:

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + L(\mathbf{w}),$$

$$L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.$$

- $L(\mathbf{w})$ is a convex, non-smooth function, with bounded Lipschitz constant (i.e., it does not vary too fast). Optimisation of such functions is extensively studied in operational research.
- We are going to discuss the Bundle Method for Regularized Risk Minimization (BMRM) method, a special case of bundle method for regularised loss functions, which in turns is a stabilised variant of cutting plane.

Subgradient and subdifferential

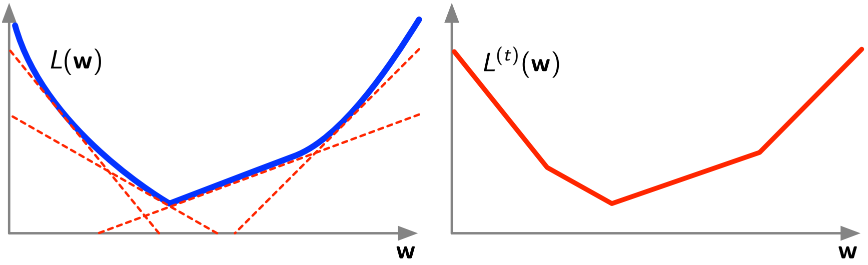


- Assumption: $L(\mathbf{w})$ convex, not necessarily smooth, with bounded Lipschitz constant G.
- A **subgradient** of $L(\mathbf{w})$ at $\mathbf{w}$ is any vector $\mathbf{g}$ such that

$$\forall \mathbf{w}' : L(\mathbf{w}') \geq L(\mathbf{w}) + \langle \mathbf{g}, \mathbf{w}' - \mathbf{w} \rangle.$$

- $\|\mathbf{g}\| \leq G$.
- The **subdifferential** $\partial L(\mathbf{w})$ is the set of all subgradients and contains only the gradient $\nabla L(\mathbf{w})$ if the function is differentiable.

Cutting planes



- Given a point $\mathbf{w}_0$, we approximate the convex $L(\mathbf{w})$ from below by a tangent plane:

$$L(\mathbf{w}) \geq b - \langle \mathbf{a}, \mathbf{w} \rangle, \quad -\mathbf{a} \in \partial L(\mathbf{w}_0) \quad b = L(\mathbf{w}_0) + \langle \mathbf{a}, \mathbf{w}_0 \rangle.$$

- $(\mathbf{a}, b)$ is the **cutting plane** at $\mathbf{w}$.
- Given the cutting planes at $\mathbf{w}_1, \ldots, \mathbf{w}_t$, we define the lower approximation

$$\boxed{L^{(t)}(\mathbf{w}) = \max_{i=1,\ldots,t} b_i - \langle \mathbf{a}_i, \mathbf{w} \rangle.}$$

Cutting plane algorithm

- Goal: minimize a convex non-necessarily smooth function $L(\mathbf{w})$.
- Method: incrementally construct a lower approximation $L^{(t)}(\mathbf{w})$. At each iteration, minimise the latter to obtain $\mathbf{w}_t$ and add a cutting plane at that point.

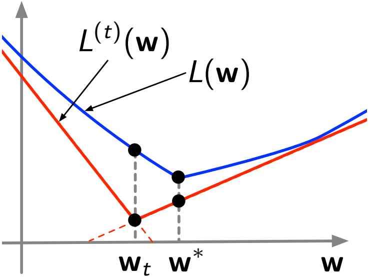
Cutting plane algorithm

Start with $\mathbf{w}_0 = 0$ and $t = 0$. Then repeat:

1. $t \leftarrow t + 1$.
2. Get a cutting plane $(\mathbf{a}_t, b_t)$ by computing the subgradient of $L(\mathbf{w})$ at $\mathbf{w}_{t-1}$.
3. Add the plane to the current approximation $L^{(t)}(\mathbf{w})$.
4. Set $\mathbf{w}_t = \operatorname{argmin}_{\mathbf{w}} L^{(t)}(\mathbf{w})$.
5. If $L(\mathbf{w}_t) - L^{(t)}(\mathbf{w}_t) < \epsilon$ stop as converged.

[Kiwiel, 1990, Lemaréchal et al., 1995, Joachims et al., 2009]

Guarantees at convergence



- The algorithm stops when $L(\mathbf{w}_t) - L^{(t)}(\mathbf{w}_t) < \epsilon$.
- The true optimum $L(\mathbf{w}^*)$ is sandwiched:

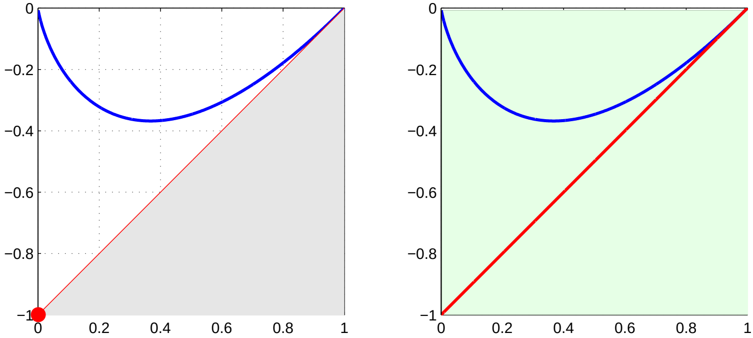
$$\overbrace{L^{(t)}(\mathbf{w}_t) \leq L^{(t)}(\mathbf{w}^*)}^{\mathbf{w}_t \text{ minimizes } L^{(t)}} \leq \overbrace{L(\mathbf{w}^*) \leq L(\mathbf{w}_t)}^{\mathbf{w}^* \text{ minimizes } L}$$

$$L^{(t)}(\mathbf{w}^*) \underbrace{\leq}_{L^{(t)} \leq L} L(\mathbf{w}^*)$$

Hence when the algorithm converge one has the guarantee:

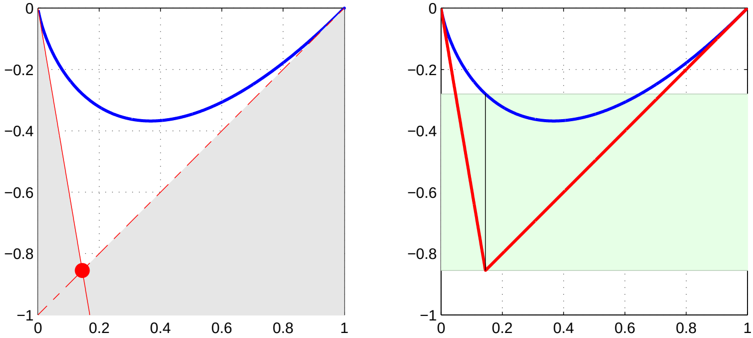
$$L(\mathbf{w}_t) \leq L(\mathbf{w}^*) + \epsilon.$$

Cutting plane example



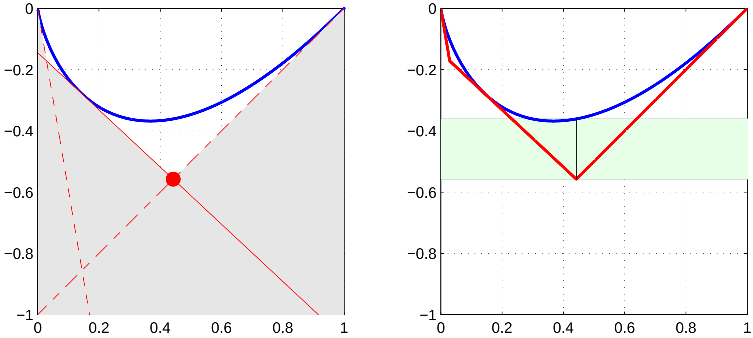
- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

Cutting plane example



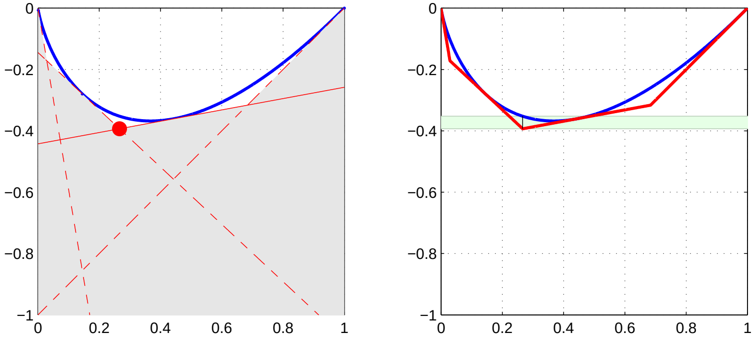
- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

Cutting plane example



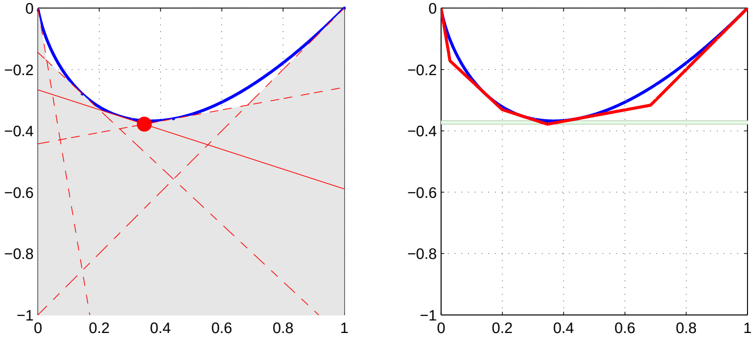
- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

Cutting plane example



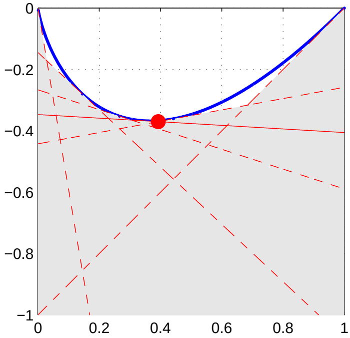
- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

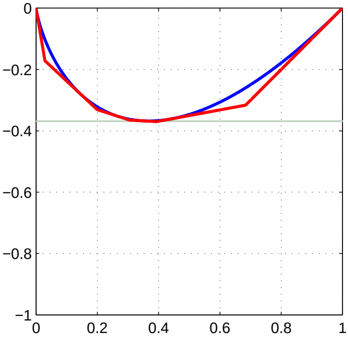
Cutting plane example



- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

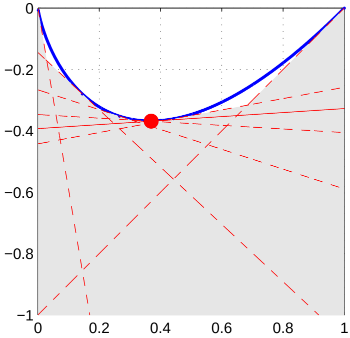
Cutting plane example

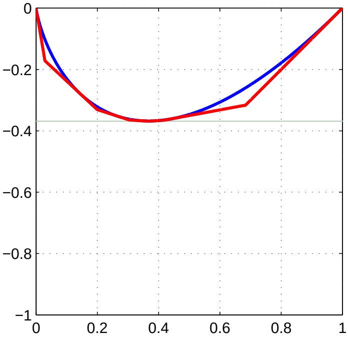




- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

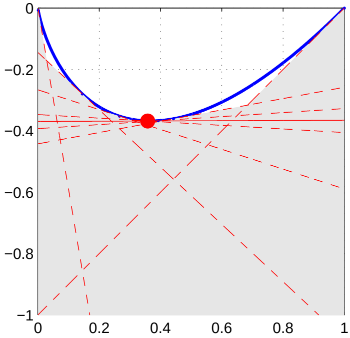
Cutting plane example

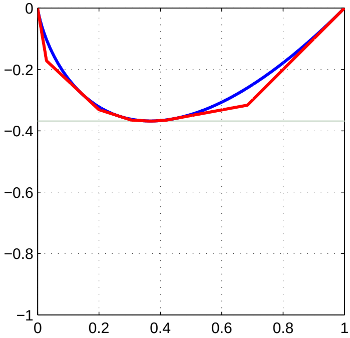




- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

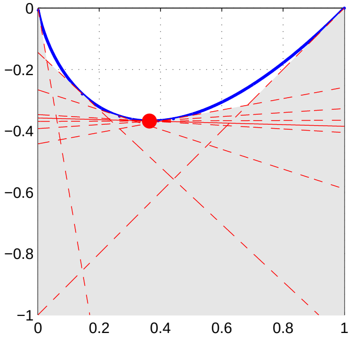
Cutting plane example

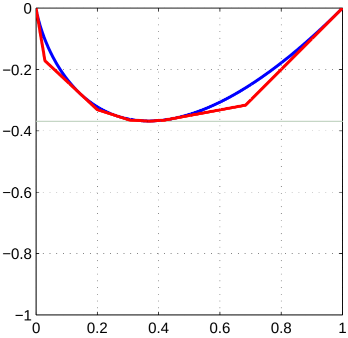




- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

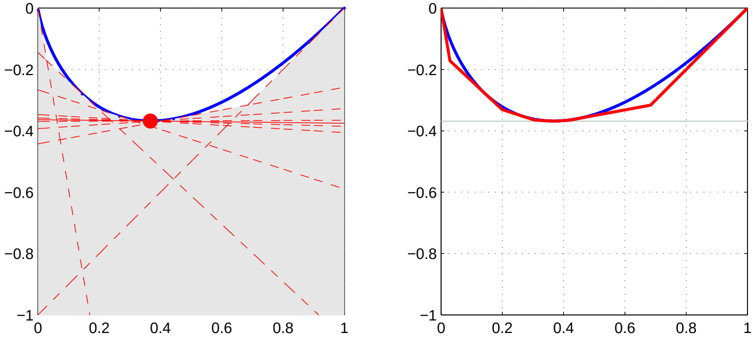
Cutting plane example





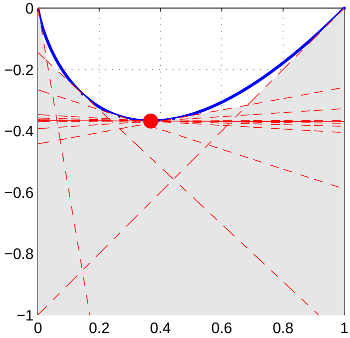
- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

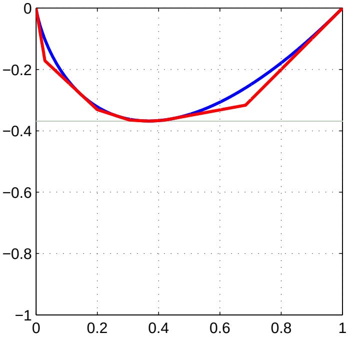
Cutting plane example



- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

Cutting plane example





- Optimizing the function $L(w) = w \log w$ in the interval $[0.001, 1]$.

BMRM: cutting planes with a regulariser

- The standard cutting plane algorithm takes forever to converge (it is *not* the one used for SVM...) as it can take wild steps.
- *Bundle methods* try to regularise the steps but are generally difficult to tune. **BMRM** notes that one has already a regulariser in the SVM objective function:

$$E(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + L(\mathbf{w}).$$

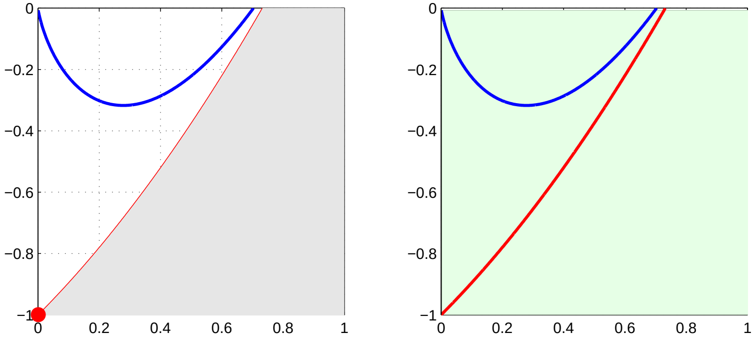
BMRM algorithm

Start with $\mathbf{w}_0 = 0$ and $t = 0$. Then repeat:

1. $t \leftarrow t + 1$.
2. Get a cutting plane $(\mathbf{a}_t, b_t)$ by computing the subgradient of $L(\mathbf{w})$ at $\mathbf{w}_{t-1}$.
3. Add the plane to the current approximation $L^{(t)}(\mathbf{w})$.
4. Set $E_t(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + L^{(t)}(\mathbf{w})$.
5. Set $\mathbf{w}_t = \operatorname{argmin}_{\mathbf{w}} E_t(\mathbf{w})$.
6. If $E(\mathbf{w}_t) - E_t(\mathbf{w}_t) < \epsilon$ stop as converged.

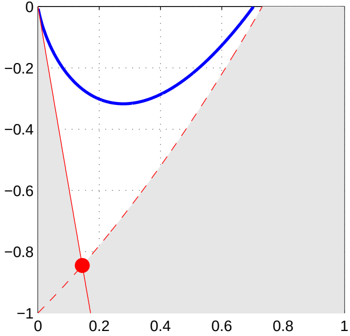
[Teo et al., 2009] but also [Kiwiel, 1990, Lemaréchal et al., 1995, Joachims et al., 2009]

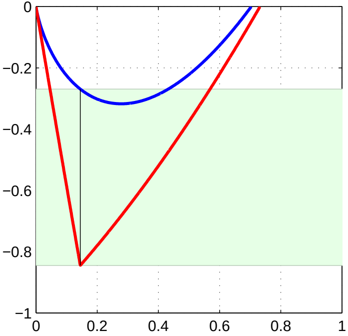
BMRM example



- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

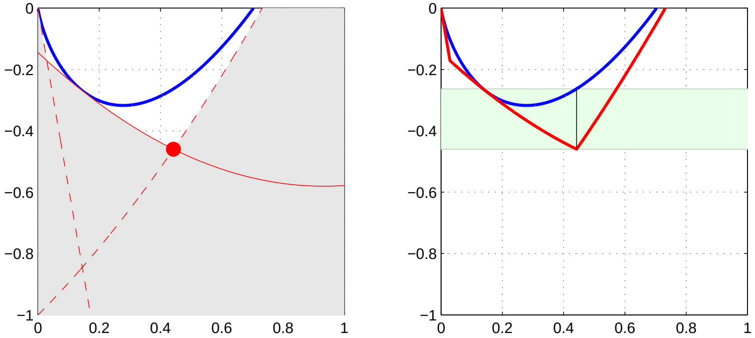
BMRM example





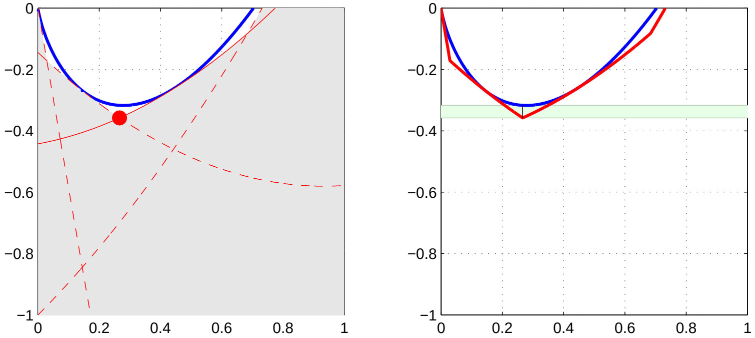
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



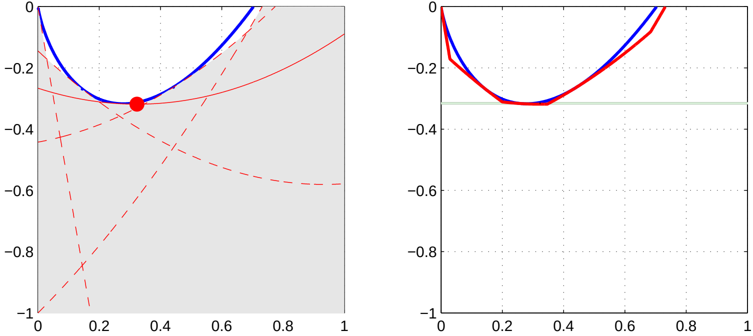
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



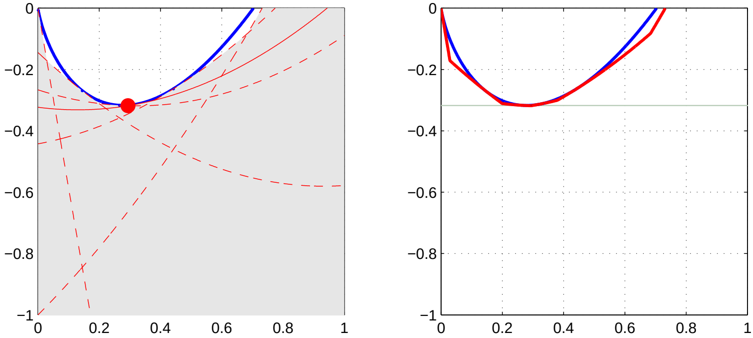
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



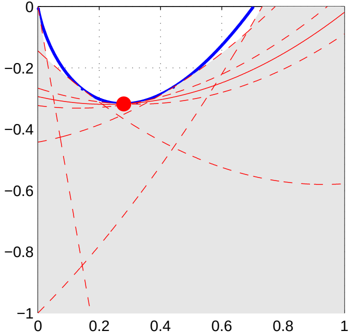
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

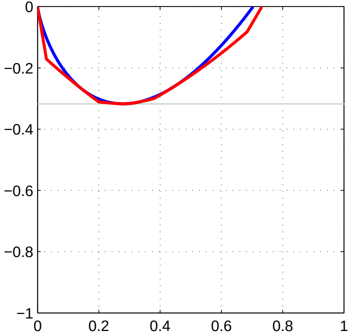
BMRM example



- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

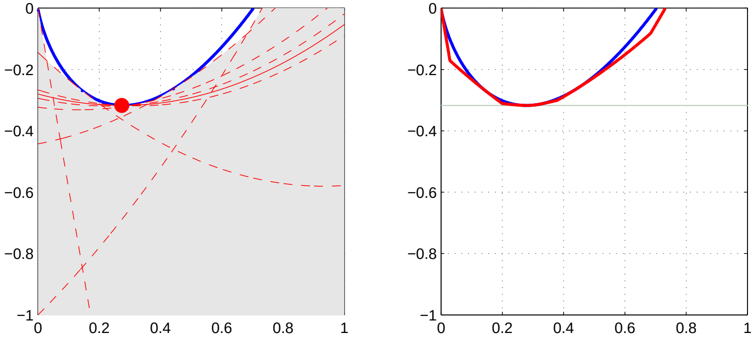
BMRM example





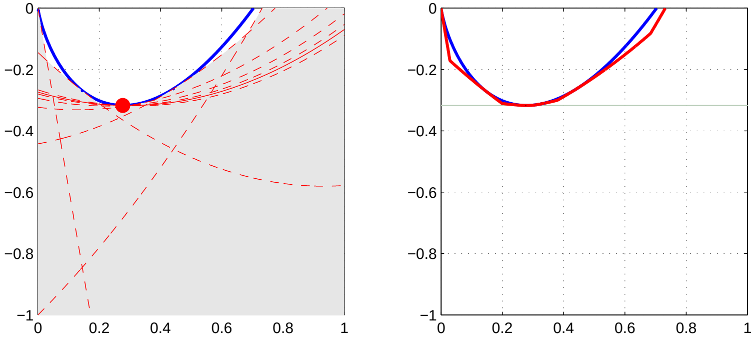
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



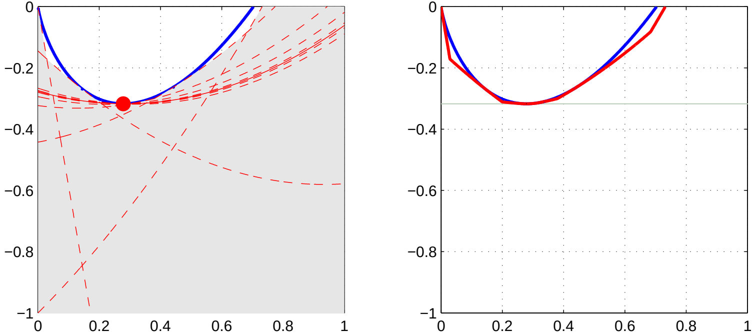
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



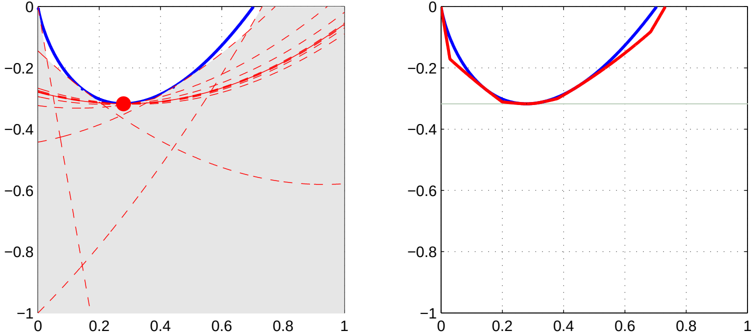
- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

BMRM example



- Optimizing the function $E(w) = \frac{w^2}{2} + w \log w$ in the interval $[0.001, 1]$.

Application of BMRM to structured SVMs

- In this case:

$$L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.$$

- $\partial L(\mathbf{w})$ is just the average of the subgradients of the terms.
- The subgradient $\mathbf{g}_i$ at $\mathbf{w}$ of a term is computed by determining the **maximally violated output**

$$\boxed{\bar{\mathbf{y}}_i = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle,}$$

Application of BMRM to structured SVMs

- In this case:

$$L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \sup_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle.$$

- $\partial L(\mathbf{w})$ is just the average of the subgradients of the terms.
- The subgradient $\mathbf{g}_i$ at $\mathbf{w}$ of a term is computed by determining the **maximally violated output**

$$\boxed{\bar{\mathbf{y}}_i = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w}\rangle - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w}\rangle,}$$

- **Remark 1.** This is the augmented inference problem.
- **Remark 2.** Once $\bar{\mathbf{y}}_i$ is obtained, the subgradient is given by

$$\mathbf{g}_i = \Psi(\mathbf{x}_i, \bar{\mathbf{y}}_i) - \Psi(\mathbf{x}_i, \mathbf{y}_i).$$

- Thus BMRM can be applied provided that the augmented inference problem can be solved (even when $\mathcal{Y}$ is infinite!).

Outline

Structured SVM: fitting a real function

- Consider the problem of learning a real function $f : \mathbb{R} \to [-1, 1]$ by fitting points $(x_1, y_1), \dots, (x_n, y_n)$.
- **Loss**

$$\Delta(y, \hat{y}) = |\hat{y} - y|.$$

- **Joint feature map**

$$\Psi(x, y) = \begin{bmatrix} y \\ yx \\ yx^2 \\ yx^3 \\ -\frac{1}{2}y^2 \end{bmatrix}.$$

To see why this works we will look at the resulting inference problem.

MATLAB implementation /1

First, program a callback for the loss.

```
function delta = lossCB(param, y, ybar)
  delta = abs(ybar - y) ;
end
```

MATLAB implementation /1

First, program a callback for the loss.

```
function delta = lossCB(param, y, ybar)
  delta = abs(ybar - y) ;
end
```

Then a callback for the feature map.

```
function psi = featureCB(param, x, y)
  psi = [y ;
         y * x ;
         y * x^2 ;
         y * x^3 ;
         - 0.5 * y^2] ;
  psi = sparse(psi) ;
end
```

Inference

- The inference problem is

$$\hat{y}(x;\mathbf{w}) = \underset{y\in[-1,1]}{\operatorname{argmax}}\langle \mathbf{w}, \Psi(x,y)\rangle$$
$$= \underset{y\in[-1,1]}{\operatorname{argmax}}\, y(w_1 + w_2 x + w_3 x^2 + w_4 x^3) - \frac{1}{2} y^2 w_5.$$

Inference

- The inference problem is

$$\hat{y}(x;\mathbf{w}) = \underset{y\in[-1,1]}{\operatorname{argmax}}\langle \mathbf{w}, \Psi(x,y)\rangle$$
$$= \underset{y\in[-1,1]}{\operatorname{argmax}}\, y(w_1 + w_2 x + w_3 x^2 + w_4 x^3) - \frac{1}{2}y^2 w_5.$$

- Differentiate w.r.t. y and set to zero to obtain:

$$\hat{y}(x;\mathbf{w}) = \frac{w_1}{w_5} + \frac{w_2}{w_5}x + \frac{w_3}{w_5}x^2 + \frac{w_4}{w_5}x^3.$$

- Note: there are some other special cases due to the fact that $y \in [-1, +1]$ and w_5 may be negative.

Augmented inference

- Solving the augmented inference problem is needed to compute the value and sub-gradient of the **margin-rescaling loss**

$$
\begin{aligned}
L_i(\mathbf{w}) &= \max_{\hat{y}\in[-1,1]} \Delta(y,\hat{y}) + \langle \mathbf{w}, \Psi(x,y)\rangle - \langle \mathbf{w}, \Psi(x,y_i)\rangle \\
&= \max_{\hat{y}\in[-1,1]} |\hat{y} - y_i| + y(w_1 + w_2 x + w_3 x^2 + w_4 x^3) - \frac{1}{2} y^2 w_5 - \text{const.}
\end{aligned}
$$

Augmented inference

- Solving the augmented inference problem is needed to compute the value and sub-gradient of the **margin-rescaling loss**

$$
\begin{aligned}
L_i(\mathbf{w}) &= \max_{\hat{y}\in[-1,1]} \Delta(y,\hat{y}) + \langle \mathbf{w}, \Psi(x,y)\rangle - \langle \mathbf{w}, \Psi(x,y_i)\rangle \\
&= \max_{\hat{y}\in[-1,1]} |\hat{y} - y_i| + y(w_1 + w_2 x + w_3 x^2 + w_4 x^3) - \frac{1}{2} y^2 w_5 - \text{const.}
\end{aligned}
$$

- The maximiser is one of at most four possibilities:

$$
y \in \left\{-1, 1, \frac{z-1}{w_5}, \frac{z+1}{w_5}\right\} \cap [-1,1], \quad z = y(w_1 + w_2 x + w_3 x^2 + w_4 x^3).
$$

- Try the four cases and pick the one with larger augmented loss.

MATLAB implementation /2

Finally program the augmented inference.

```
function yhat = constraintCB(param, model, x, y)
  w = model.w ;
  z = w(1) + w(2) * x + w(3) * x.^2 + w(4) * x.^3 ;
  yhat = [] ;
  if w(5) > 0
    yhat = [z - 1, z + 1] / w(5) ;
    yhat = max(min(yhat, 1),-1) ;
  end
  yhat = [yhat, -1, 1] ;

  aloss = @(y_) abs(y_ - y) + z * y_ - 0.5 * y_.^2 * w(5) ;
  [drop, worse] = max(aloss(yhat)) ;
  yhat = yhat(worse) ;
end
```

MATLAB implementation /3

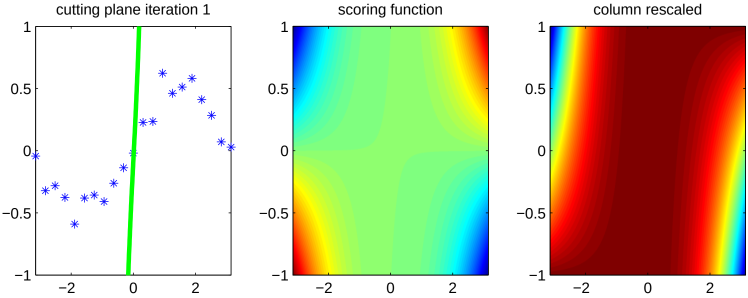
Once the callbacks are coded, we use an off-the-shelf-solver (`http://www.vlfeat.org/~vedaldi/code/svm-struct-matlab.html`)

```
% training examples
parm.patterns = {-2, -1, 0, 1, 2} ;
parm.labels = {0.5, -0.5, 0.5, -0.5, 0.5} ;

% callbacks & other parameters
parm.lossFn = @lossCB ;
parm.constraintFn = @constraintCB ;
parm.featureFn = @featureCB ;
parm.dimension = 5 ;

% call the solver and print the model
model = svm_struct_learn(' -c 10 -o 2 ', parm) ;
model.w
```

Learning the scoring function



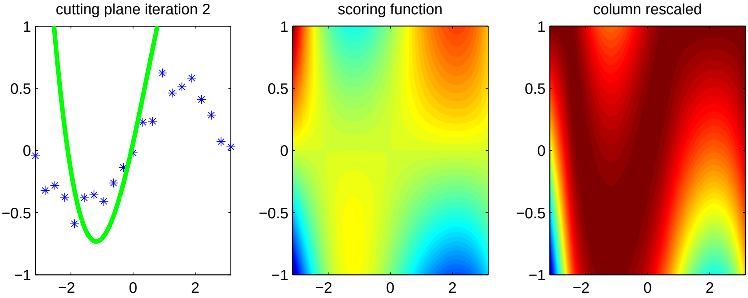
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



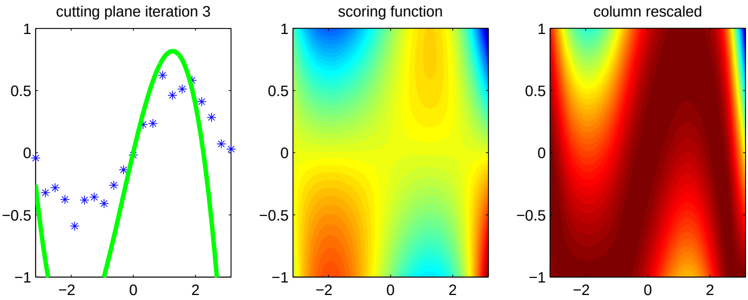
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



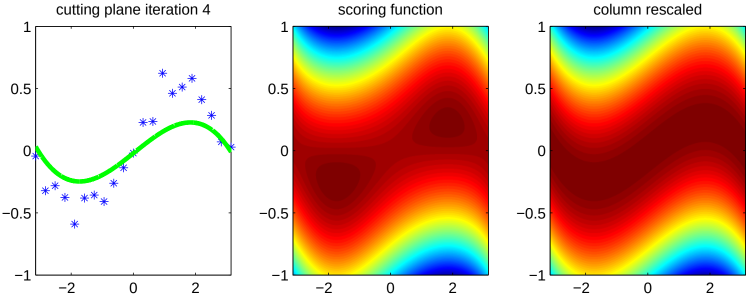
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



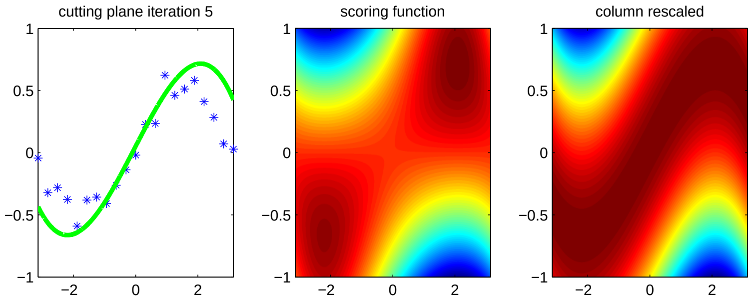
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



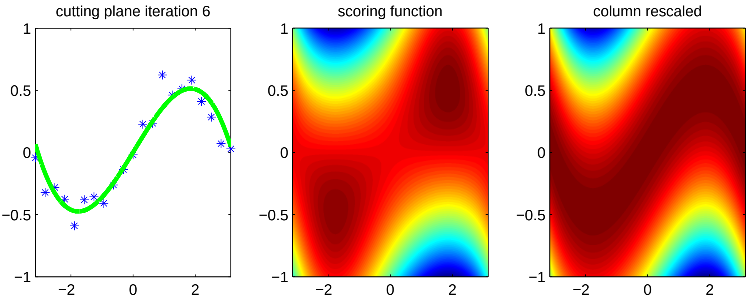
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



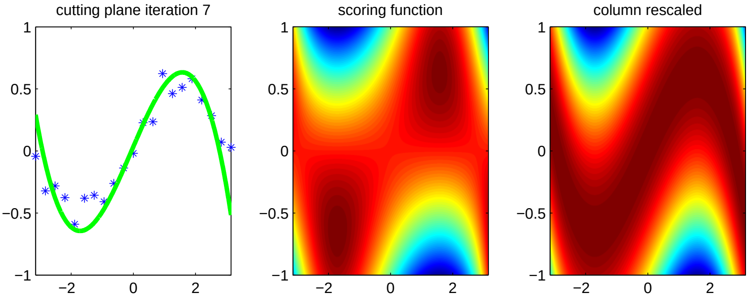
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



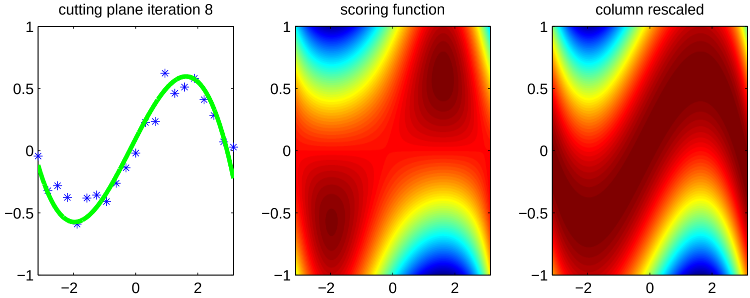
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



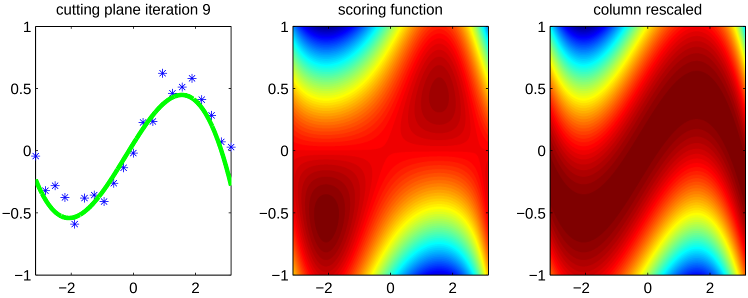
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



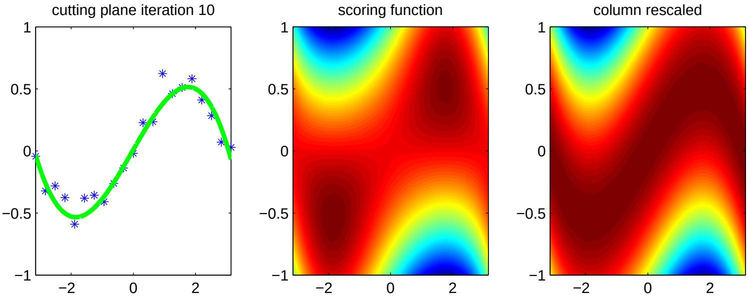
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



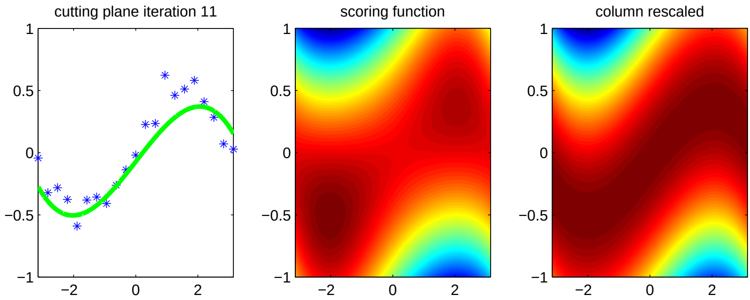
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



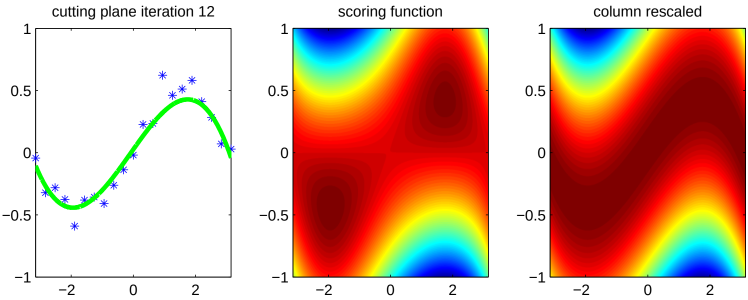
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



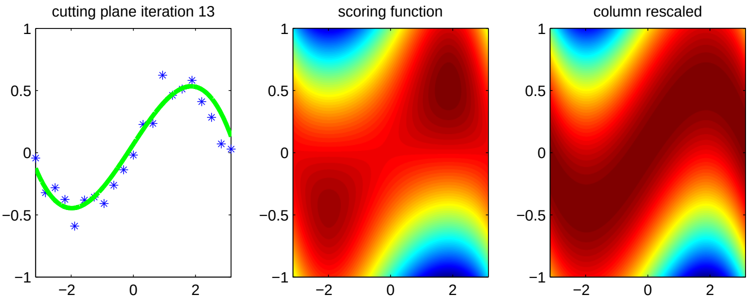
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



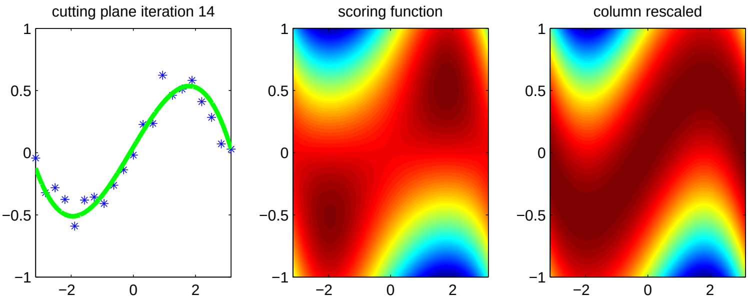
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



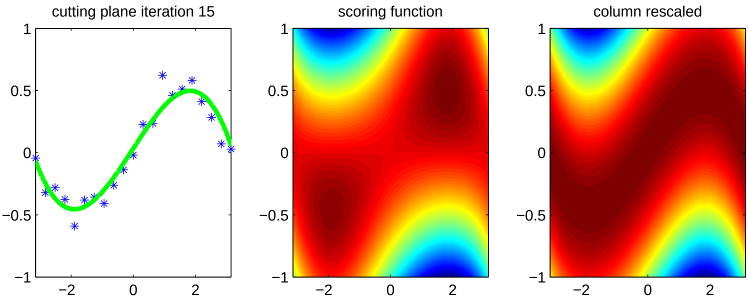
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



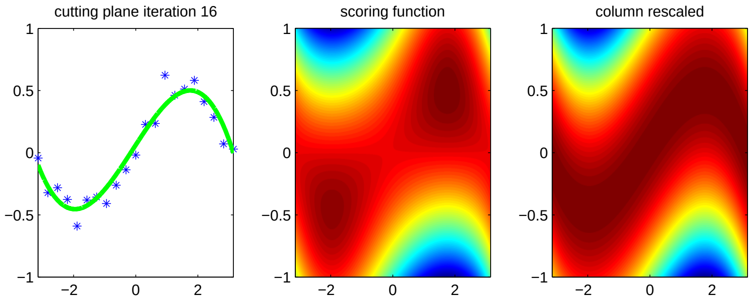
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



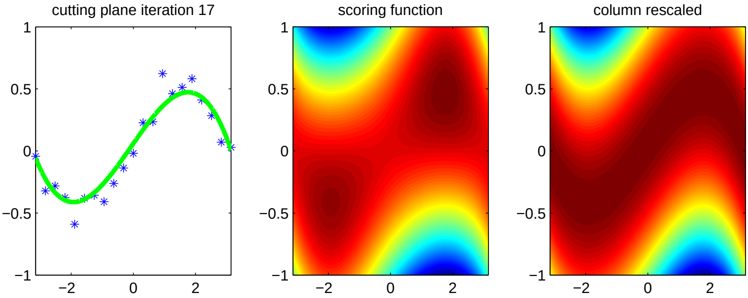
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



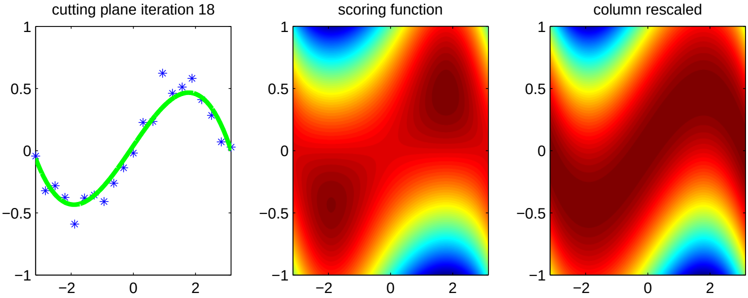
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



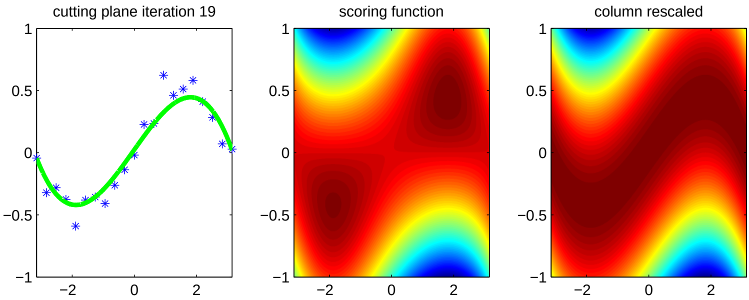
After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Learning the scoring function



After each cutting plane iteration the scoring function

$$F(x, y) = \langle \Psi(x, y), \mathbf{w} \rangle$$

is updated. Remember:

- The output function is obtained by maximising the score along the columns.
- The relative scaling of the columns is irrelevant and rescaling them reveals the structure better.

Outline

How fast is BMRM?

- Provably convergent to a desired approximation ϵ.
- The convergence rates with respect to the accuracy ϵ are not bad:

loss $L(\mathbf{w})$	number of iterations	accounting for λ
non-smooth	$O(\frac{1}{\epsilon})$	$O(\frac{1}{\lambda\epsilon})$
smooth	$O(\log(\frac{1}{\epsilon}))$	$O(\frac{1}{\lambda}\log(\frac{1}{\epsilon}))$

- Note: the convergence rate depends *also* on the amount of regularisation λ.
- Difficult learning problems (e.g. object detection) typically have
  - large n,
  - small λ,
  - small ϵ.

  so fast convergence is not so obvious.

BMRM for structured SVMs: problem size

- BMRM *decouples* the data from the approximation of $L(\mathbf{w})$.
- The number of data points n affects the cost of evaluationg $L(\mathbf{w})$ and its subgradient.
- However, the cost of optimising $L^{(t)}(\mathbf{w})$ depends only on the iteration number t!
- In practice t is small and $L^{(t)}(\mathbf{w})$ may be minimised very efficiently in the dual.

BMRM subproblem in the primal

- The problem

$$\min_{\mathbf{w}} \frac{\lambda}{2}\|\mathbf{w}\|^2 + L^{(t)}(\mathbf{w}), \quad L^{(t)}(\mathbf{w}) = \max_{i=1,\ldots,n} b_i - \langle \mathbf{a}_i, \mathbf{w} \rangle$$

  reduces to the constrained quadratic program

$$\min_{\mathbf{w},\xi} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \xi,$$
$$\xi \geq b_i - \langle \mathbf{a}_i, \mathbf{w} \rangle, \quad i = 1, \ldots, t.$$

- Note that there is a single (scalar) slack variable. This is known as **one-slack formulation**.

BMRM subproblem in the dual

Let $\mathbf{b}^\top = [b_1, \ldots, b_t]$, $A = [\mathbf{a}_1, \ldots, \mathbf{a}_t]$ and $K = A^\top A/\lambda$. The corresponding dual problem is

$$\max_{\boldsymbol{\alpha} \geq 0} \langle \boldsymbol{\alpha}, \mathbf{b} \rangle - \frac{1}{2} \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}, \quad \mathbf{1}^\top \boldsymbol{\alpha} \leq 1.$$

where at optimum $\mathbf{w}^* = \frac{1}{\lambda} A \boldsymbol{\alpha}^*$.

Intuition: why it is efficient

- The original infinite constraints are approximated by just t constraints in $L^{(t)}(\mathbf{w})$.
- This is possible because:
  1. The approximation needs to be good only around the optimum.
  2. The effective dimensionality and redundancy of the data are exploited.
- Solving the corresponding quadratic problem is easy because t is small.

Remark. BMRM is a primal solver. Switching to the dual for the subproblems is convenient but completely optional.

Implementation

- An attractive aspect is the ease of implementation.

```
A = [] ;
B = [] ;
minimum = -inf ;
while getObjective(w) - minimum > epsilon
  [a,b] = getCuttingPlane(w) ;
  A = [A, a] ;
  B = [B, b] ;
  [w, minimum] = quadraticSolver(lambda, A, B) ;
end
```

- A simple quadratic solver may do as the problem is small (e.g. MATLAB `quadprog`).
- `getCuttingPlane` computes an average of subgradients, in turn obtained by solving the augmented inference problems.

Tricks of the trade: caching /1

	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\dots$
$L_1(\mathbf{w})$	$(\mathbf{a}_{11}, b_{11})$	$(\mathbf{a}_{12}, b_{12})$	$(\mathbf{a}_{13}, b_{13})$	$\dots$
$L_2(\mathbf{w})$	$(\mathbf{a}_{21}, b_{21})$	$(\mathbf{a}_{22}, b_{22})$	$(\mathbf{a}_{23}, b_{23})$	$\dots$
$\vdots$				
$L_n(\mathbf{w})$	$(\mathbf{a}_{n1}, b_1)$	$(\mathbf{a}_{n2}, b_{n2})$	$(\mathbf{a}_{n3}, b_{n3})$	$\dots$
$L(\mathbf{w})$	$(\mathbf{a}_1, b_1)$	$(\mathbf{a}_2, b_2)$	$(\mathbf{a}_3, b_3)$	$\dots$

- For each novel $\mathbf{w}_t$ a new constraint per example is generated by running augmented inference.
- The overall loss is an average of per-example losses:

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} L_i(\mathbf{w})$$

- And so for each cutting plane:

$$\mathbf{a}_t = \frac{1}{n} \sum_{i=1}^{n} \mathbf{a}_{it}(\mathbf{w}), \quad b_t = \frac{1}{n} \sum_{i=1}^{n} b_{it}(\mathbf{w}),$$

Tricks of the trade: caching /2

	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$	$\dots$	
$L_1(\mathbf{w})$	$(\mathbf{a}_{11}, b_{11})$	$(\mathbf{a}_{12}, b_{12})$	$(\mathbf{a}_{13}, b_{13})$	$\dots$	$\to t_1^*$
$L_2(\mathbf{w})$	$(\mathbf{a}_{21}, b_{21})$	$(\mathbf{a}_{22}, b_{22})$	$(\mathbf{a}_{23}, b_{23})$	$\dots$	$\to t_2^*$
$\vdots$					$\vdots$
$L_n(\mathbf{w})$	$(\mathbf{a}_{n1}, b_1)$	$(\mathbf{a}_{n2}, b_{n2})$	$(\mathbf{a}_{n3}, b_{n3})$	$\dots$	$\to t_n^*$
$L(\mathbf{w})$	$(\mathbf{a}_1, b_1)$	$(\mathbf{a}_2, b_2)$	$(\mathbf{a}_3, b_3)$	$\dots$	$(\mathbf{a}_{t+\delta t}, b_{t+\delta t})$

Caching recombines constraints generated so far to obtain a novel cutting plane without running augmented inference (expensive) [Joachims, 2006, Felzenszwalb et al., 2008].

1. For each example $i = 1, \dots, n$ pick the most violated constraint in the cache:

$$t_i^* = \operatorname*{argmax}_{t=1,\dots,t} b_{it} - \langle \mathbf{a}_{it}, \mathbf{w} \rangle.$$

2. Now form a novel cutting plane by recombining the existing constraints:

$$\mathbf{a}_{t+\delta t} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{a}_{it_i^*}(\mathbf{w}), \quad b_{t+\delta t} = \frac{1}{n} \sum_{i=1}^{n} b_{it_i^*}(\mathbf{w}), .$$

Tricks of the trade: caching /3

- Caching is **very important** for problems like object detection in which inference is very expensive (seconds or minutes per image).
- Consider for example [Felzenszwalb et al., 2008] object detector . With 5000 training images and five seconds / image for inference it requires an hour for *one* round of augmented inference!
- Thus the solver should be iterated until examples in the cache are correctly separated. It is pointless to fetch more before the solution has stabilised due to the huge cost.
- **Preventive caching.** During a round of inference it is also possible to return and store in the cache a small set of “highly violated” constraints. They may become “most violated” at a later iteration.

Tricks of the trade: incremental training

- Another speedup is to train the model gradually, by adding progressively more training samples.
- The intuition is that a lot of samples are only needed to refine the model.

Summary

- Structured output SVMs extend standard SVMs to arbitrary output spaces.
- "New" optimisation techniques allow to learn models on a large scale (e.g. BMRM).
- Benefits
  - Apply well understood convex formulations and optimisation techniques to a variety problems, from ranking to image segmentation and pose estimation.
  - Good solvers + explicit feature maps = large scale non-linear models.
  - Can incorporate latent variables (not discussed).
- Caveats
  - Surrogate losses have many limitations.
  - Inference and augmented inference must be solved ad hoc for every new design.
  - All the limitations of discriminative learning.

Slides and code at

`http://www.vlfeat.org/~vedaldi/teach.html`.

Bibliography I

G. Csurka, C. R. Dance, L. Dan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proc. ECCV Workshop on Stat. Learn. in Comp. Vision*, 2004.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. CVPR*, 2009.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshop*, 2004.

P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proc. CVPR*, 2008.

V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *Proc. CVPR*, 2008.

T. Hastie, R. Tibishirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

T. Joachims. Training linear SVMs in linear time. In *Proc. KDD*, 2006.

Bibliography II

T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1), 2009.

K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46, 1990.

B. Leibe and B. Schiele. Scale-invariant object categorization using a scale-adaptive mean-shift search. *Lecture Notes in Computer Science*, 3175, 2004.

C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Mathematical Programming*, 69, 1995.

V. Lempitsky, A. Vedaldi, and A. Zisserman. A pylon model for semantic segmentation. In *Proc. NIPS*, 2011.

T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *Proc. ECCV*, 2012.

D. Parikh and K. Grauman. Relative attributes. In *Proc. ICCV*, 2011.

D. Ramanan. Learning to parse images of articulated bodies. In *Proc. NIPS*, 2006.

Bibliography III

D. Ramanan, D. Forsyth, and A. Zisserman. Strike a pose: Tracking people by finding stylized poses. In *Proc. CVPR*, 2005.

J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. CVPR*, 2011.

B. Schölkopf and A. Smola. *Learning with Kernels*, chapter Robust Estimators, pages 75 – 83. MIT Press, 2002a.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002b.

J. Sivic and A. Zisserman. Efficient visual content retrieval and mining in videos. *Lecture Notes in Computer Science*, 3332, 2004.

Alex J. Smola and Bernhard Scholkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3), 2004.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Proc. NIPS*, 2003.

C. H. Teo, S. V. N. Vishwanathan, A. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 1(55), 2009.

Bibliography IV

A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *Proc. ICCV*, 2009.

M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *Proc. CVPR*, volume 2, pages 101–108, 2000.