Learning data discretization via convex programming

Vojtěch Franc[1], Ondřej Fikar[1], Karel Bartoš[2], Michal Sofka[2]

[1]Czech Technical University in Prague
[2]Cisco Systems, Inc.

November 25, 2016

Example: Linear embeding for non-linear classification

Input features: $\boldsymbol{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$

Embedding: The original features are discretized to obtain a sparse representation $\boldsymbol{\phi}\colon \mathbb{R}^n \to \{0,1\}^{n \cdot D}$ such that

$$\phi_{ij}(\boldsymbol{x}; \boldsymbol{\nu}) = \begin{cases} 1 & \text{if } x_i \in [\nu_{i,j-1}, \nu_{i,j}) \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{\nu} = (\nu_{1,0}, \ldots, \nu_{1,D}, \ldots, \nu_{n,0}, \ldots, \nu_{n,D}) \in \mathbb{R}^{n \cdot (D+1)}$ is a set of thresholds.

Example: Linear embeding for non-linear classification

Input features: $\boldsymbol{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$

Embedding: The original features are discretized to obtain a sparse representation $\boldsymbol{\phi}\colon \mathbb{R}^n \to \{0,1\}^{n \cdot D}$ such that

$$\phi_{ij}(\boldsymbol{x};\boldsymbol{\nu}) = \begin{cases} 1 & \text{if } x_i \in [\nu_{i,j-1}, \nu_{i,j}) \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{\nu} = (\nu_{1,0}, \ldots, \nu_{1,D}, \ldots, \nu_{n,0}, \ldots, \nu_{n,D}) \in \mathbb{R}^{n \cdot (D+1)}$ is a set of thresholds.

Decision rule $h\colon \mathbb{R}^n \to \{+1,-1\}$ based on thresholding a linear score $h(\boldsymbol{x};\boldsymbol{v},\boldsymbol{\nu}) = \operatorname{sgn}(f(\boldsymbol{x};\boldsymbol{v},\boldsymbol{\nu}))$ where

$$f(\boldsymbol{x};\boldsymbol{v},\boldsymbol{\nu}) = \sum_{i=1}^{n} \sum_{j=1}^{D} v_{ij}\phi_{ij}(\boldsymbol{x};\boldsymbol{\nu})$$

Problem: How to learn $\boldsymbol{v}$ and $\boldsymbol{\nu}$?

Example: Learning weights and discretization simultaneously

Idea: construct initial discretization $\boldsymbol{\nu}$ uniformly with a high number of bins D and then merge the bins during the course of learning.

Example: Learning weights and discretization simultaneously

Idea: construct initial discretization $\boldsymbol{\nu}$ uniformly with a high number of bins D and then merge the bins during the course of learning.

Modified SVM algorithm: Given a set of training examples $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in (\mathbb{R}^n \times \{-1, 1\})^m$, learning of weights $\boldsymbol{v}$ is formulated as a convex problem:

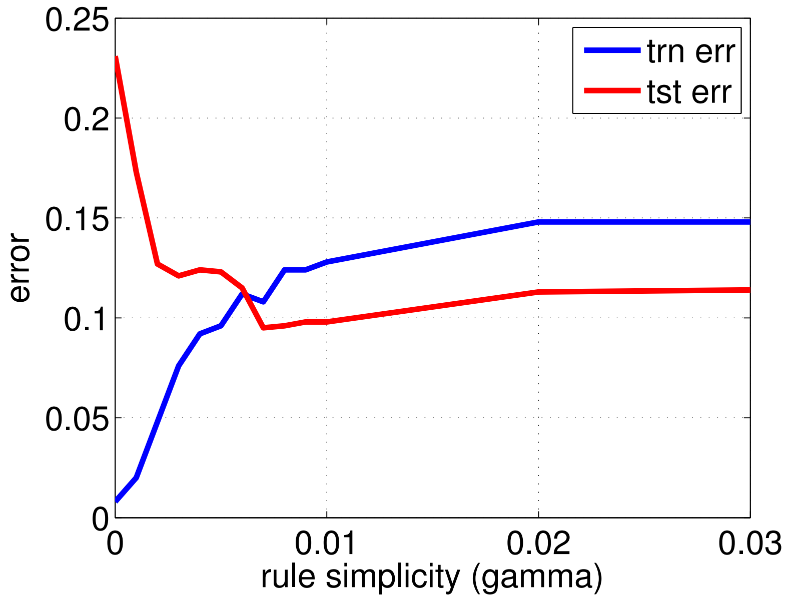
$$\min_{\boldsymbol{v} \in \mathbb{R}^{n \cdot D}} \Bigg[\underbrace{\lambda \|\boldsymbol{v}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max\Big\{0, 1 - y_i \sum_{i=1}^{n} \sum_{j=1}^{D} v_{ij} \phi_{ij}(\boldsymbol{x}; \boldsymbol{\nu})\Big\}}_{\text{SVM objective function}} + \underbrace{\gamma \sum_{i=1}^{n} \sum_{j=1}^{D-1} |v_{i,j} - v_{i,j+1}|}_{\text{Added term}} \Bigg]$$

where the hyper-parameter $\gamma > 0$ implicitly controls the number of similar weights.

Remark: $v_{i,j} = v_{i,j+1}$ is the same like merging corresponding bin $[\nu_{i,j-1}, \nu_{i,j})$ and $[\nu_{i,j}, \nu_{i,j+1})$ to a single bin $[\nu_{i,j-1}, \nu_{i,j+1})$.

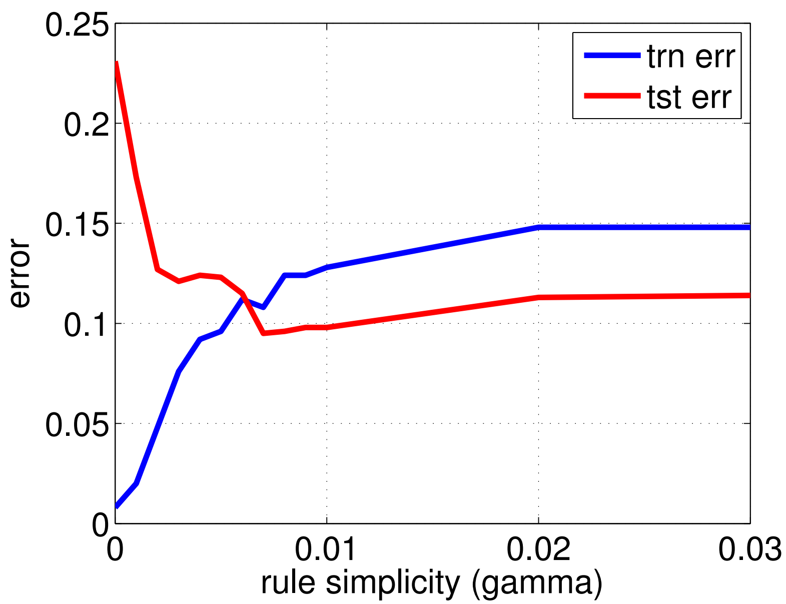
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.

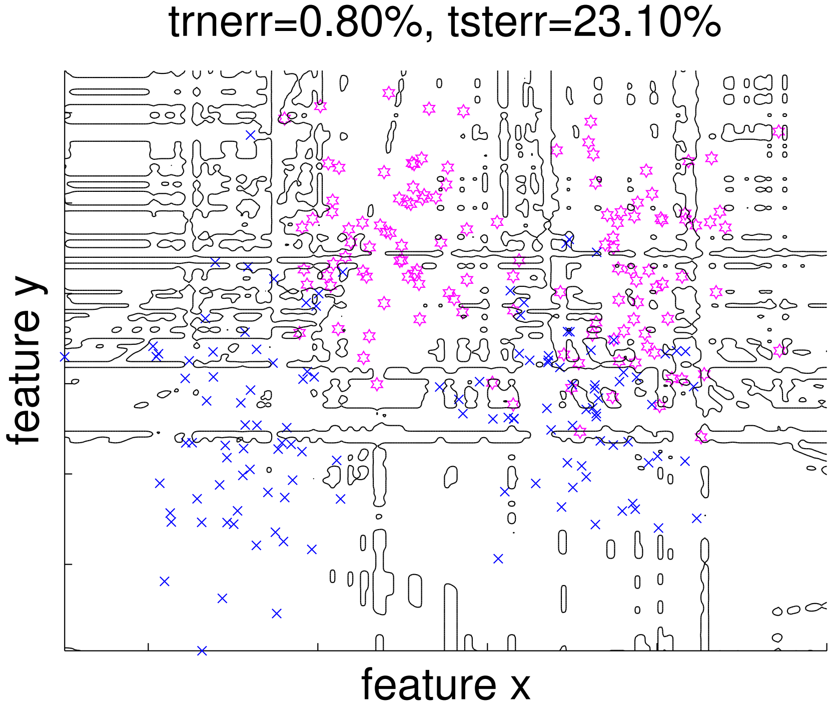


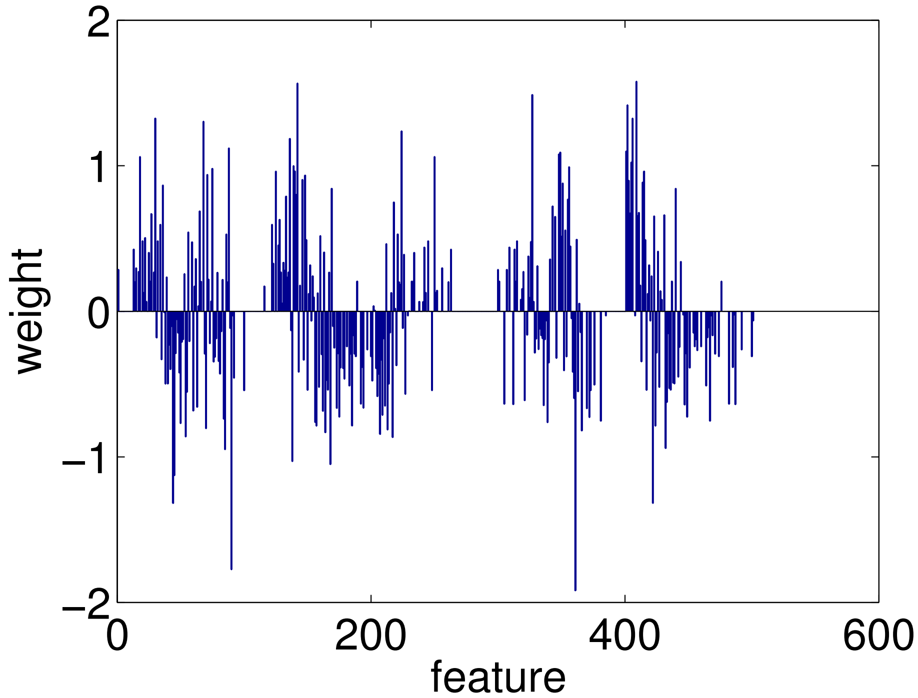
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



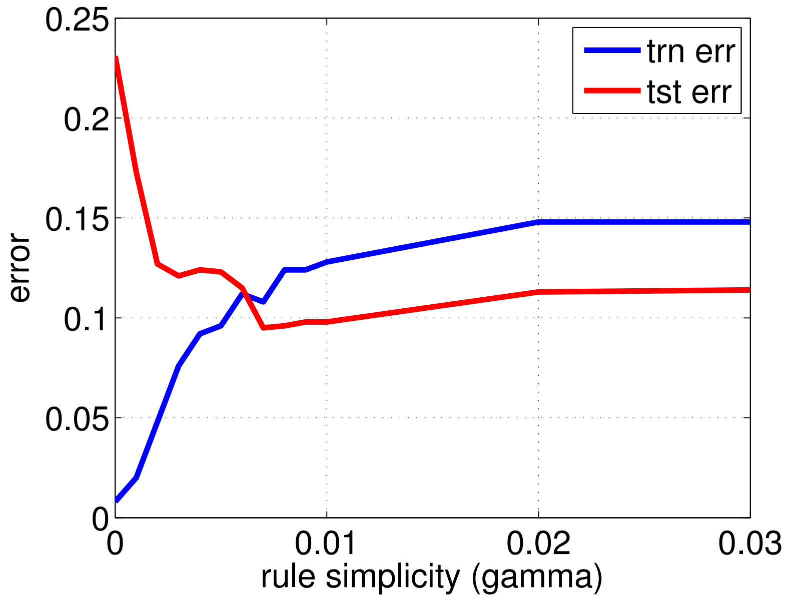
$$\lambda = 0.001,\ \gamma = 0.000$$



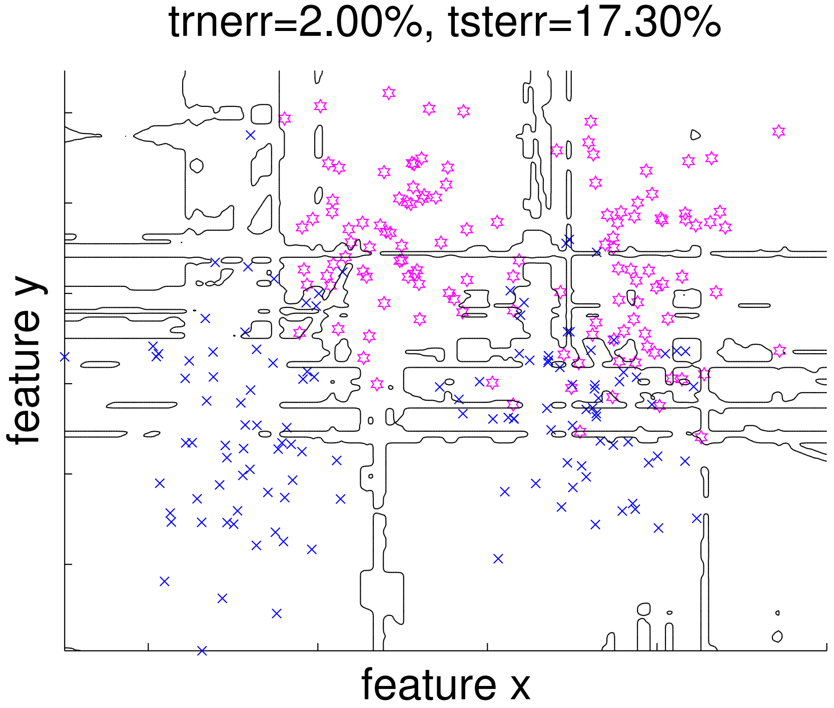


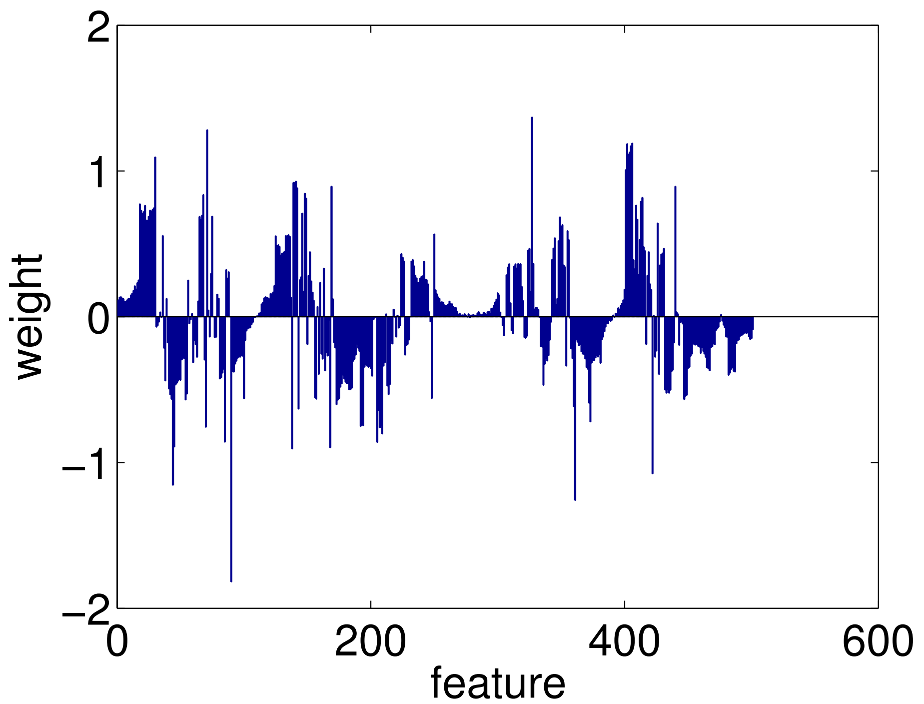
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



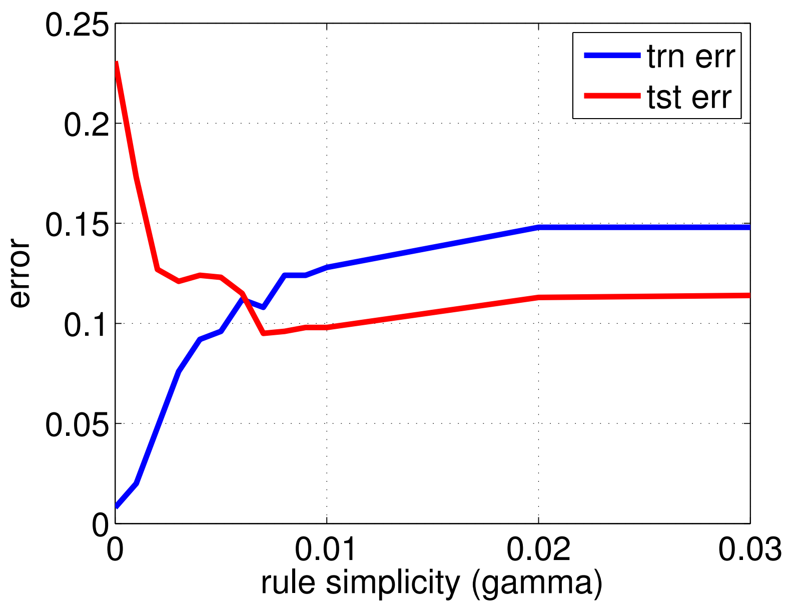
$$\lambda = 0.001, \; \gamma = 0.001$$



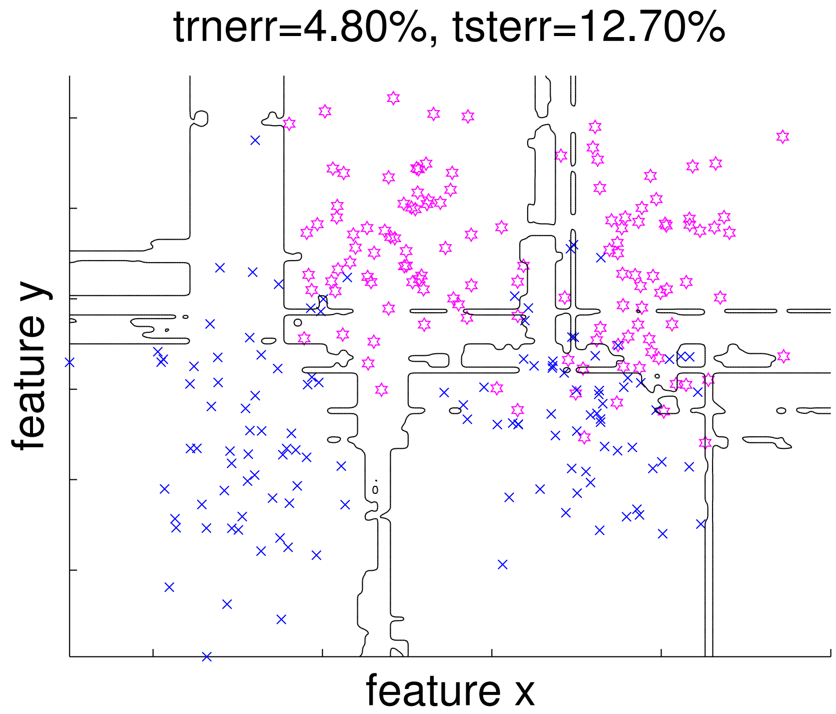


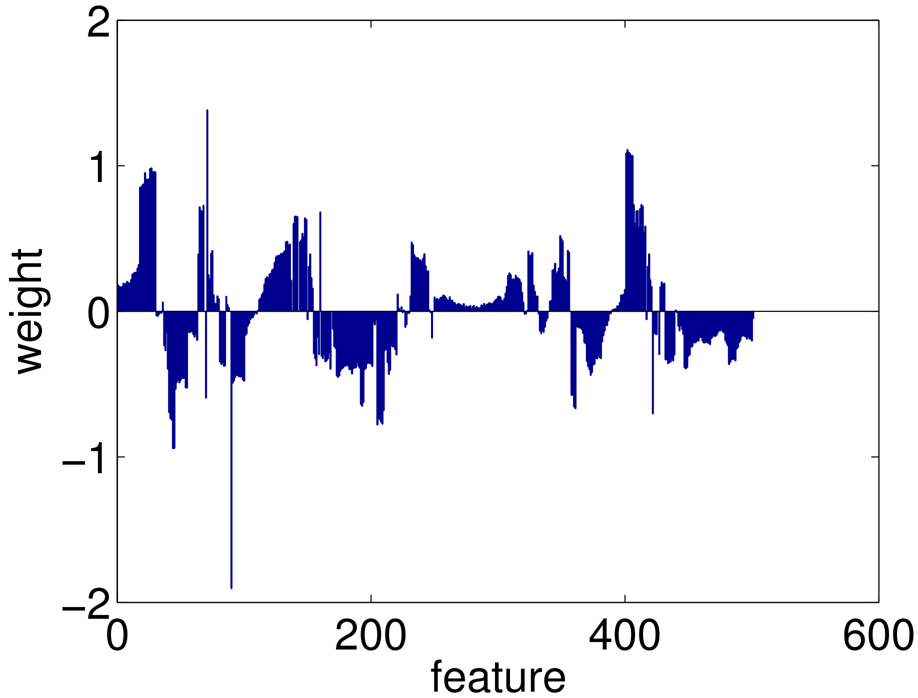
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



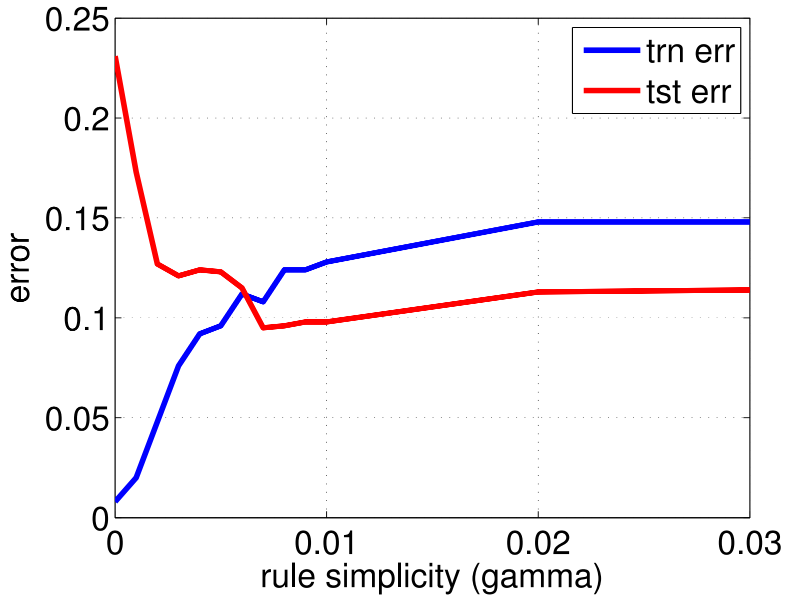
$$\lambda = 0.001, \; \gamma = 0.002$$



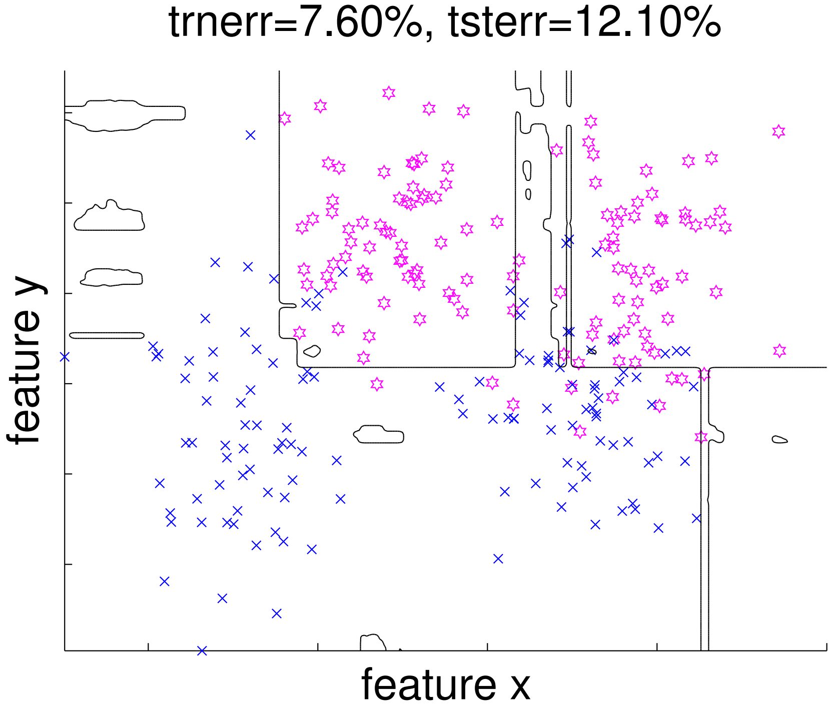


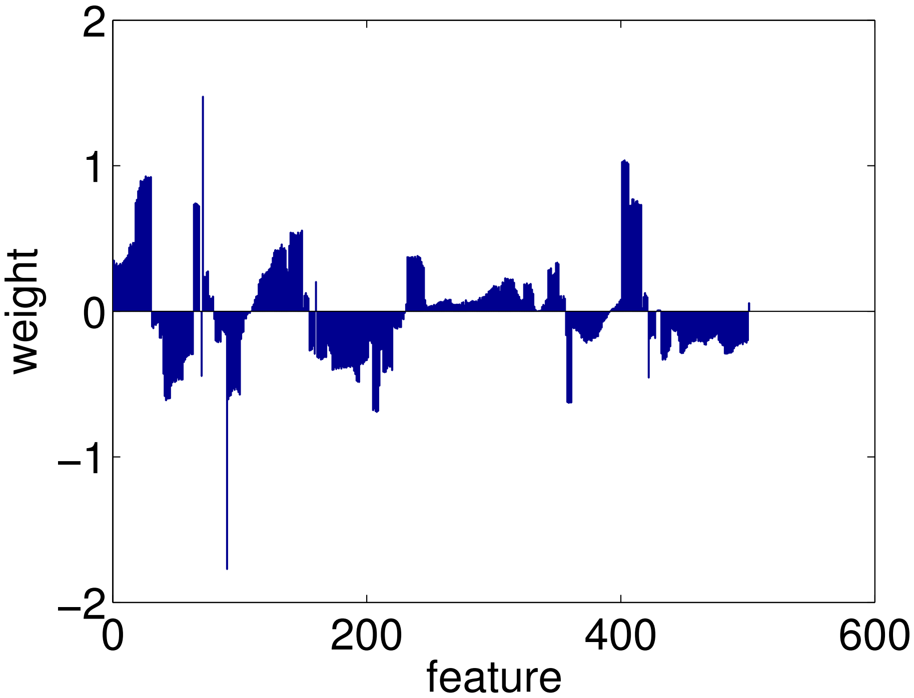
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



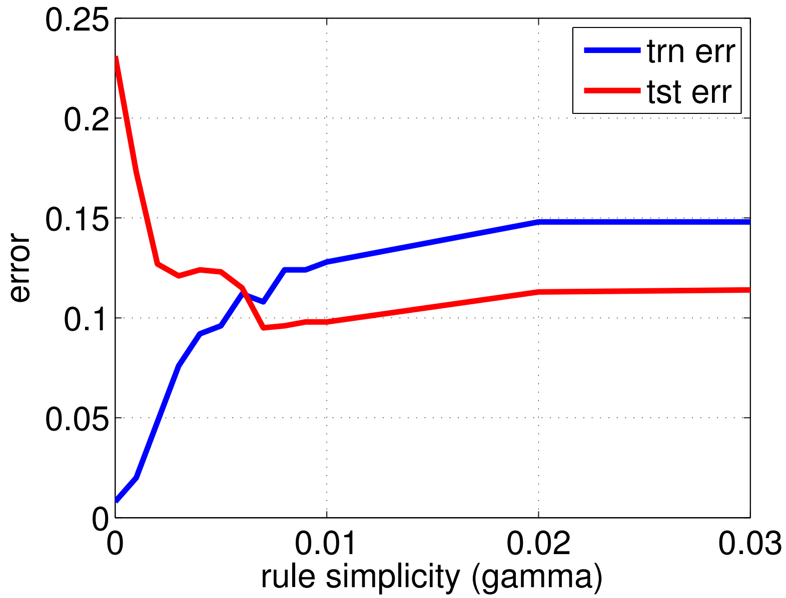
$$\lambda = 0.001, \; \gamma = 0.003$$



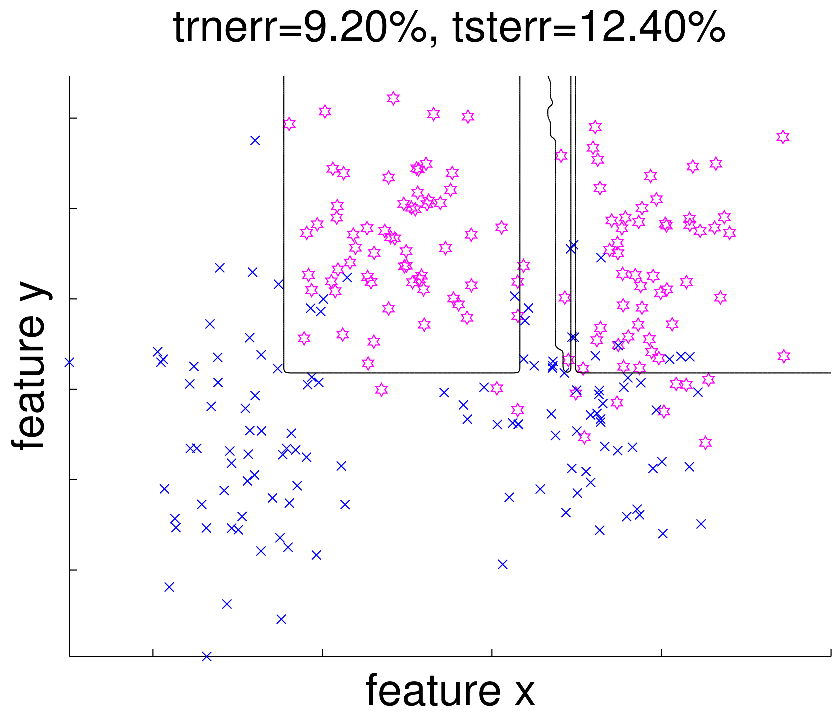


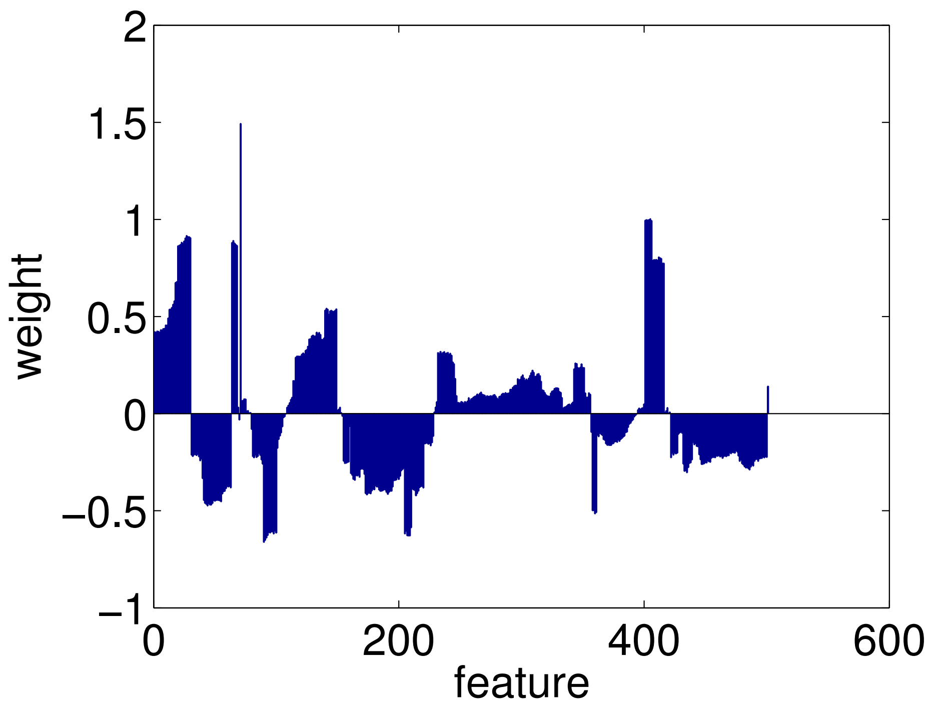
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



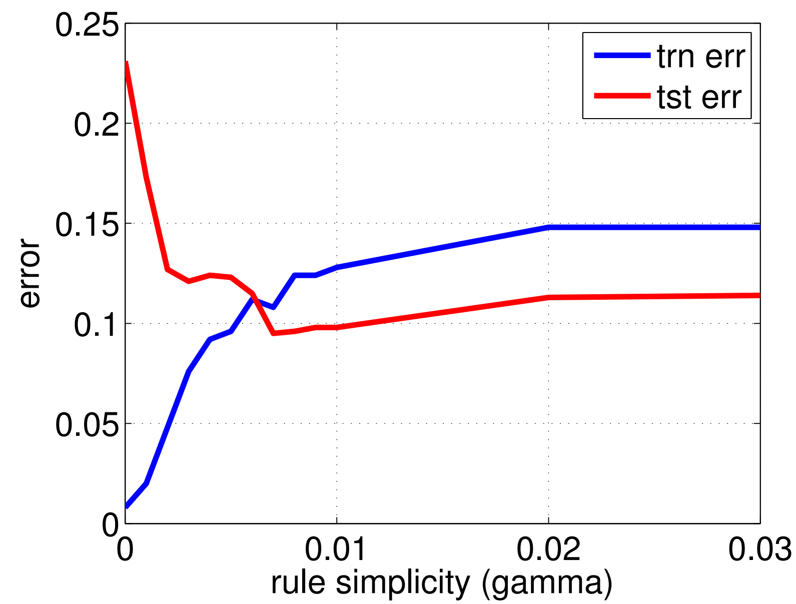
$$\lambda = 0.001, \ \gamma = 0.004$$



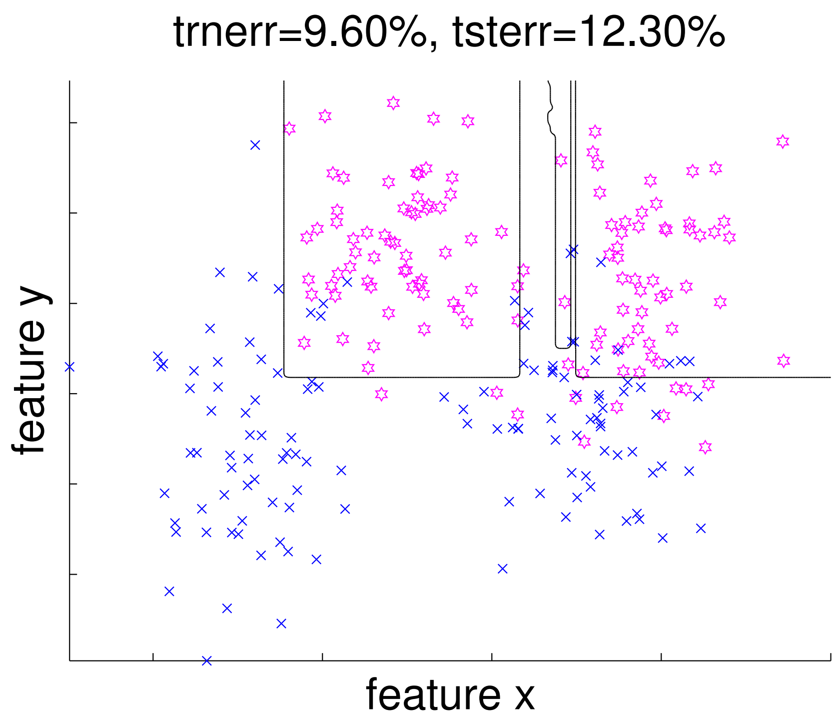


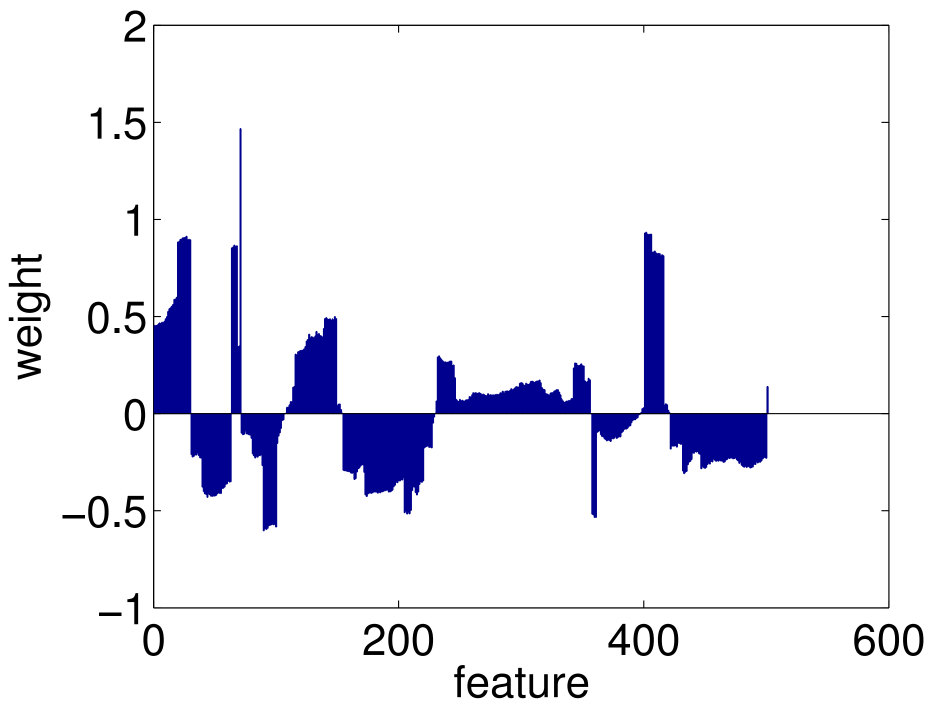
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



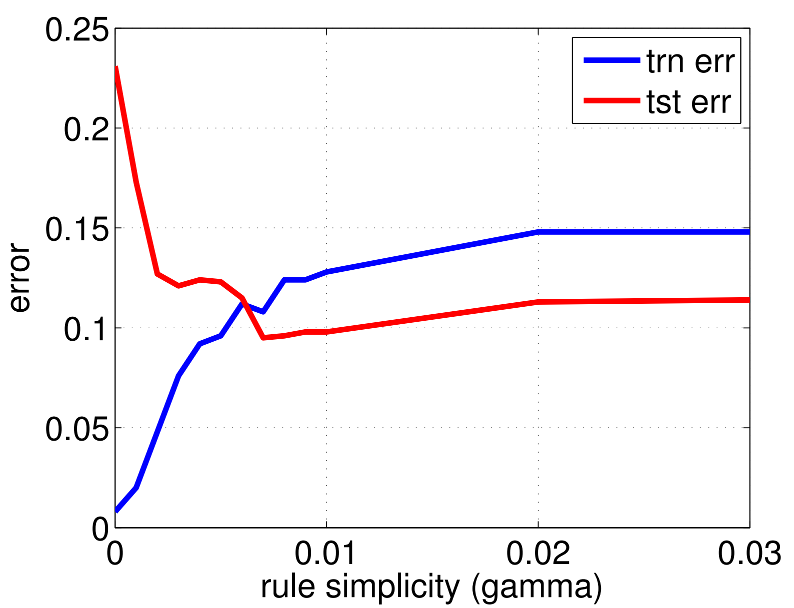
$\lambda = 0.001, \, \gamma = 0.005$



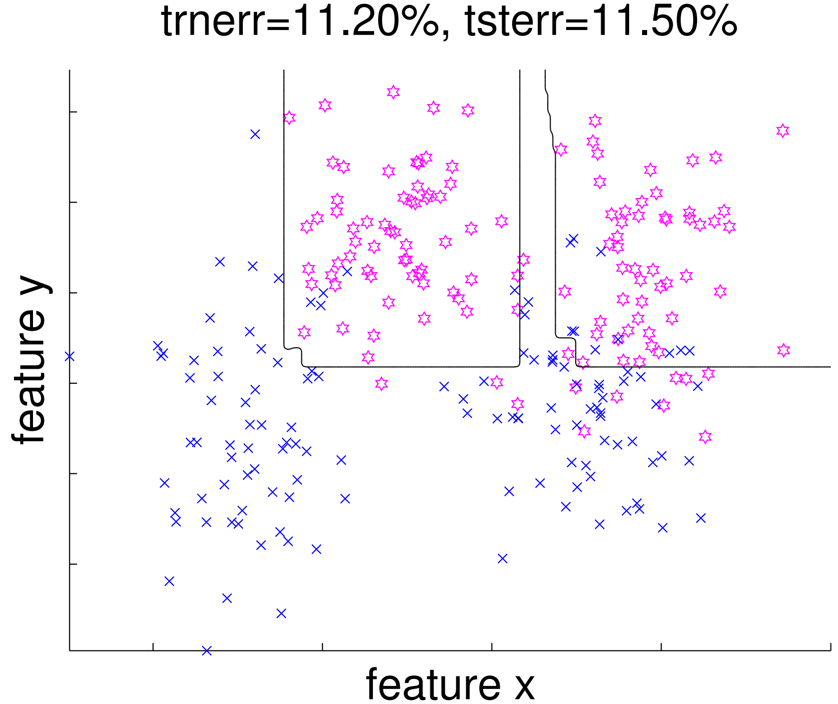


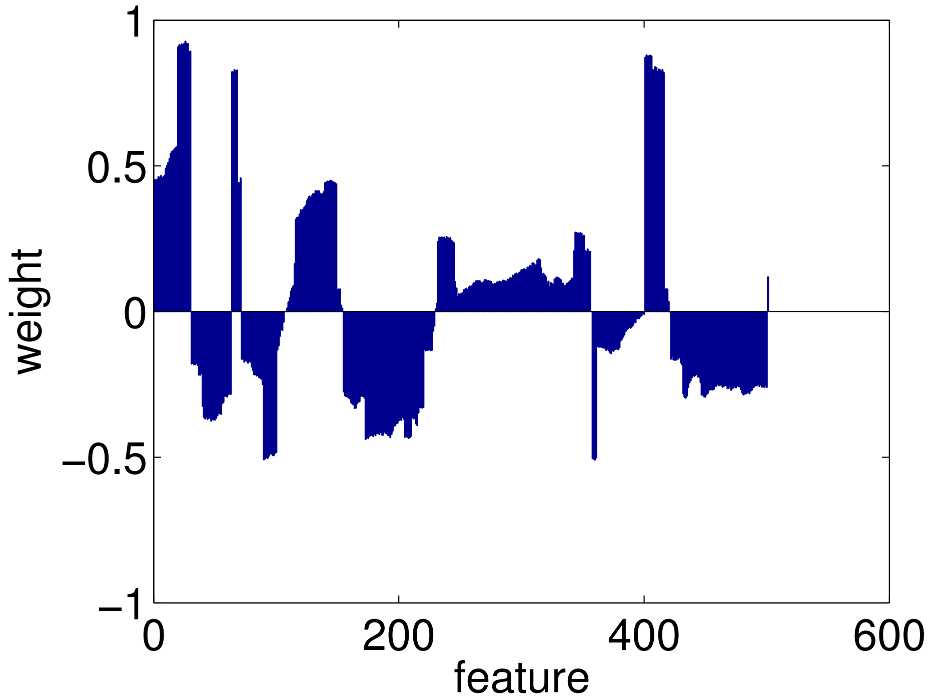
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



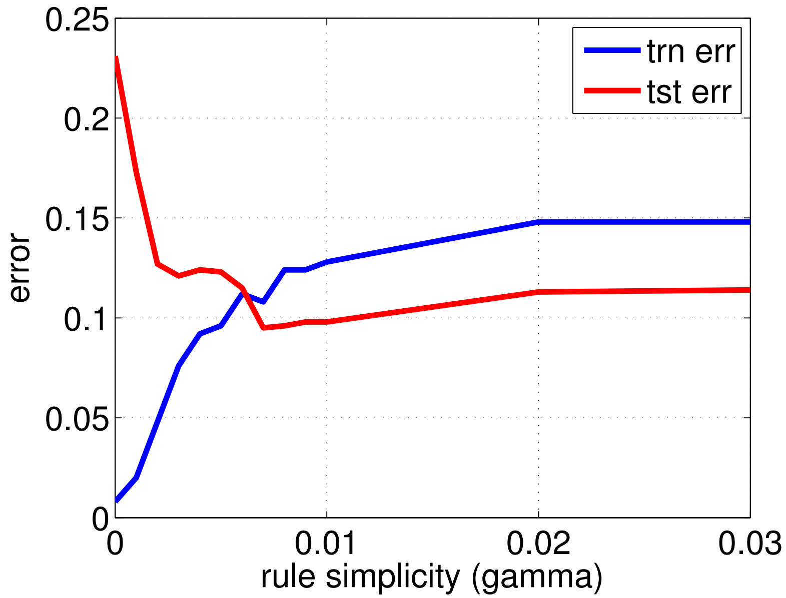
$$\lambda = 0.001,\ \gamma = 0.006$$



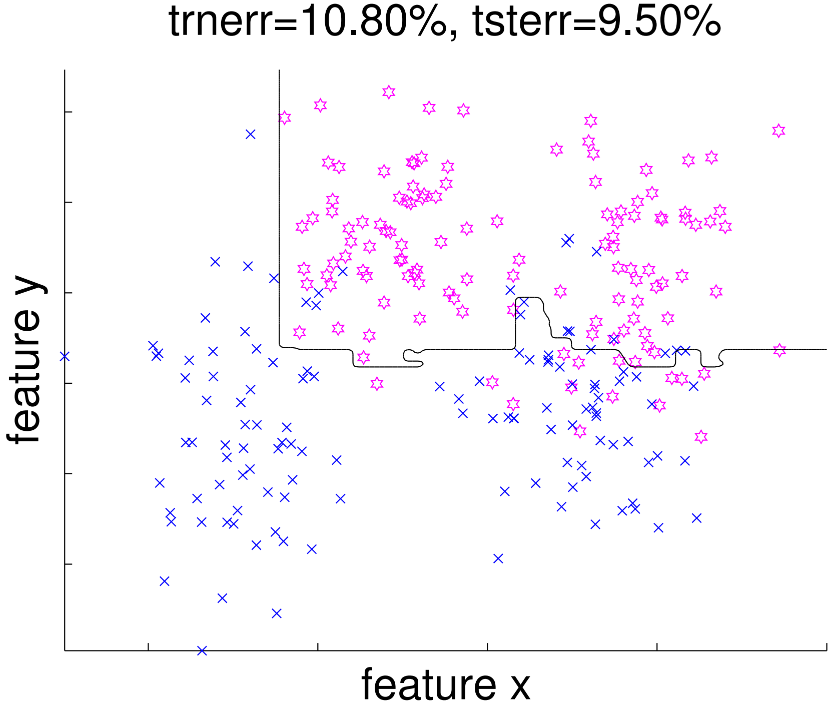


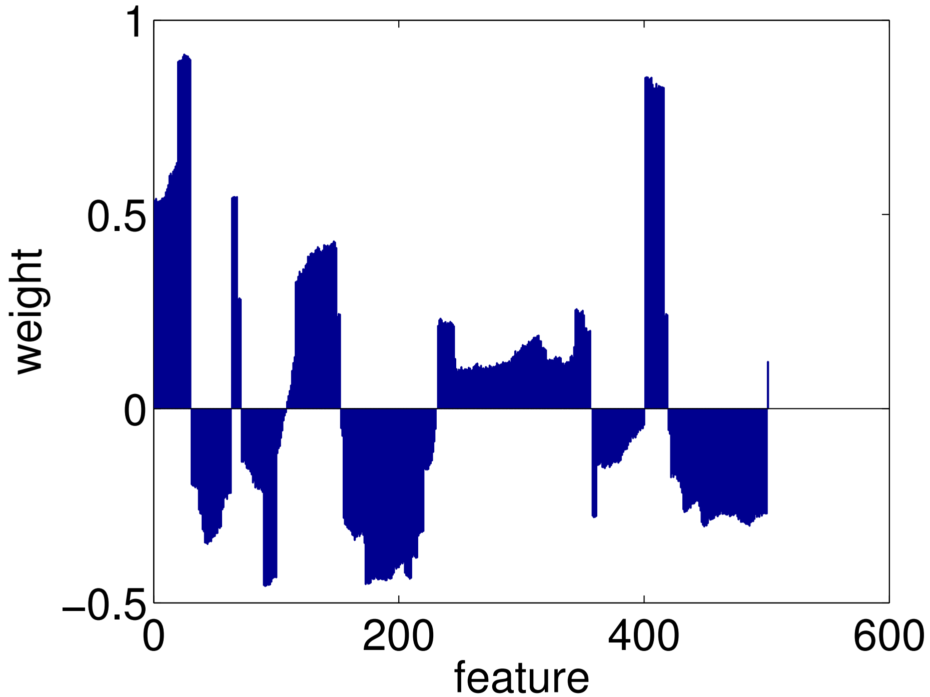
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



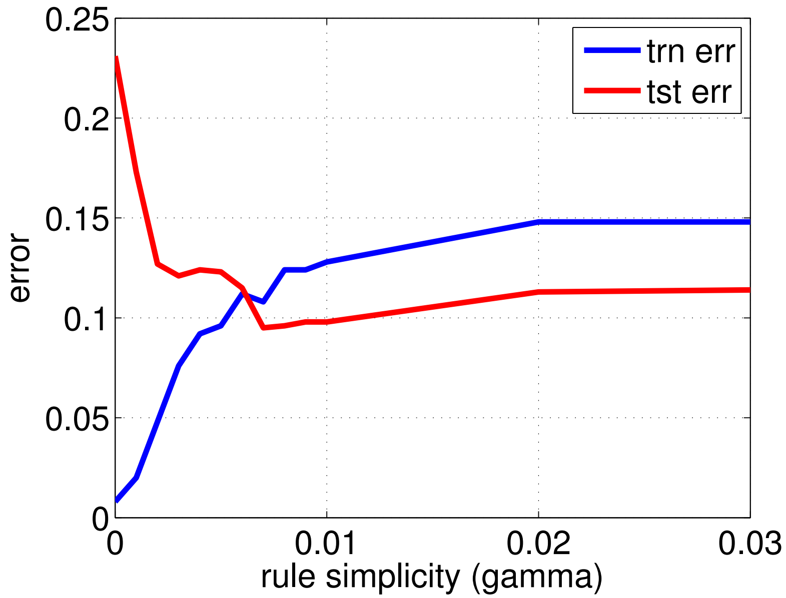
$$\lambda = 0.001,\ \gamma = 0.007$$



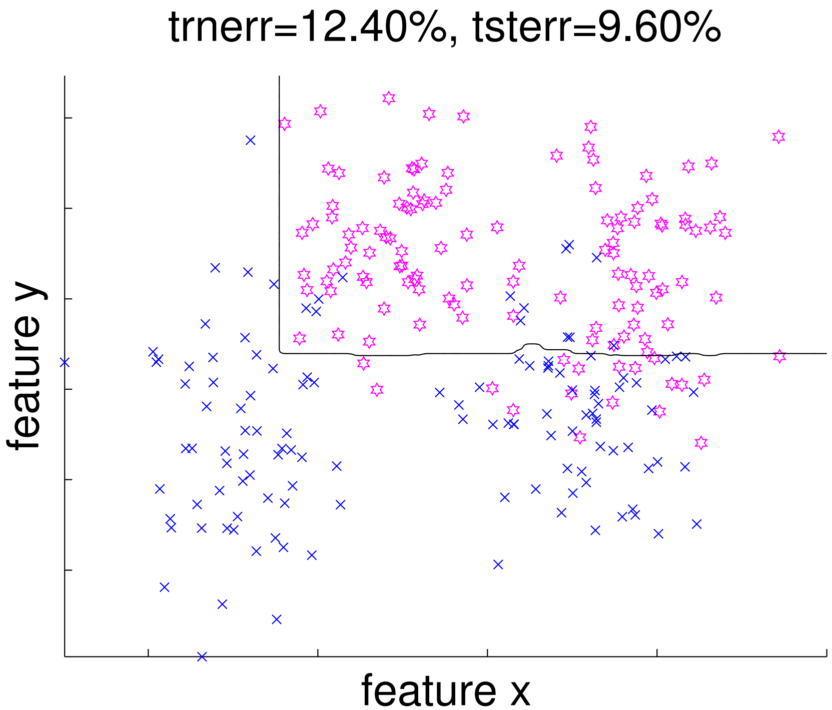


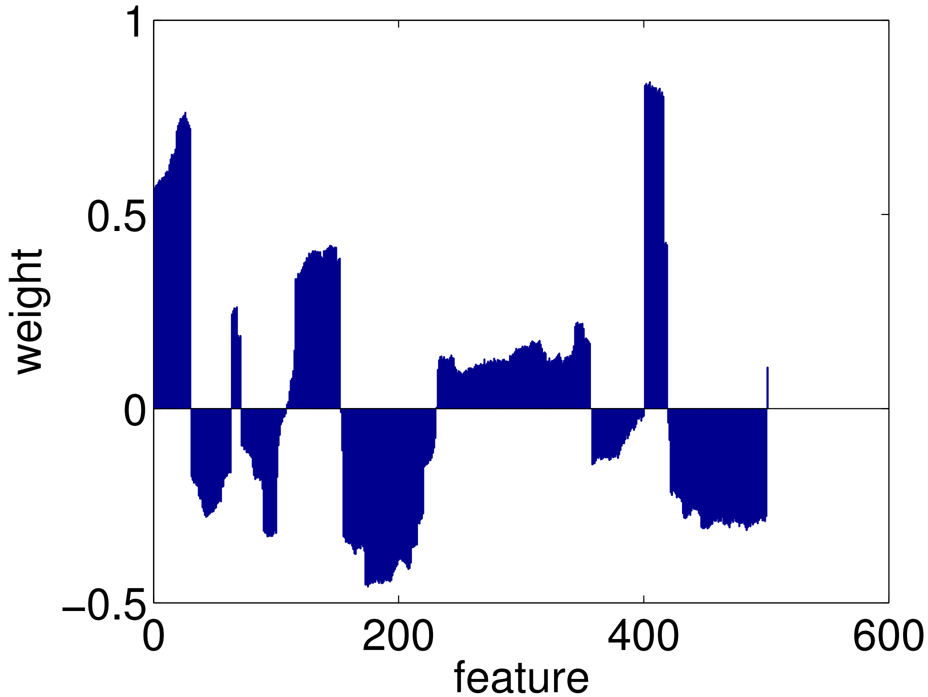
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



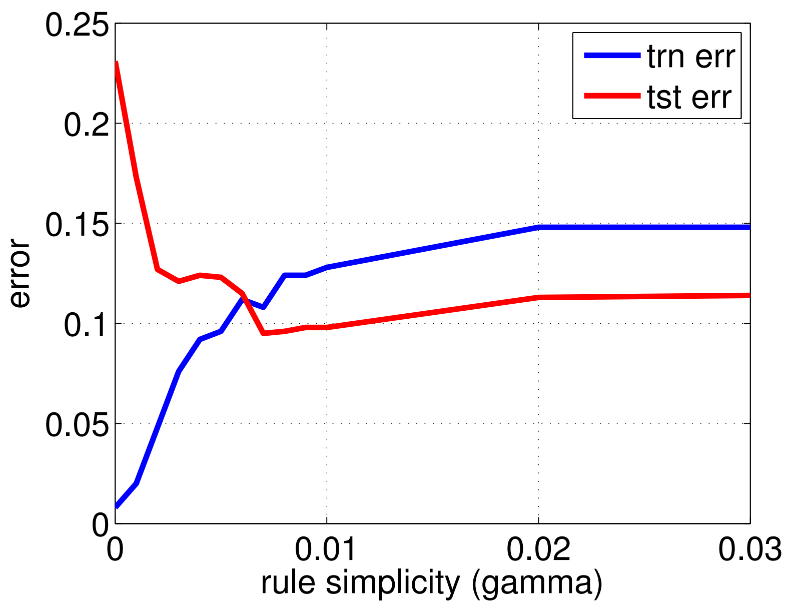
$\lambda = 0.001,\ \gamma = 0.008$



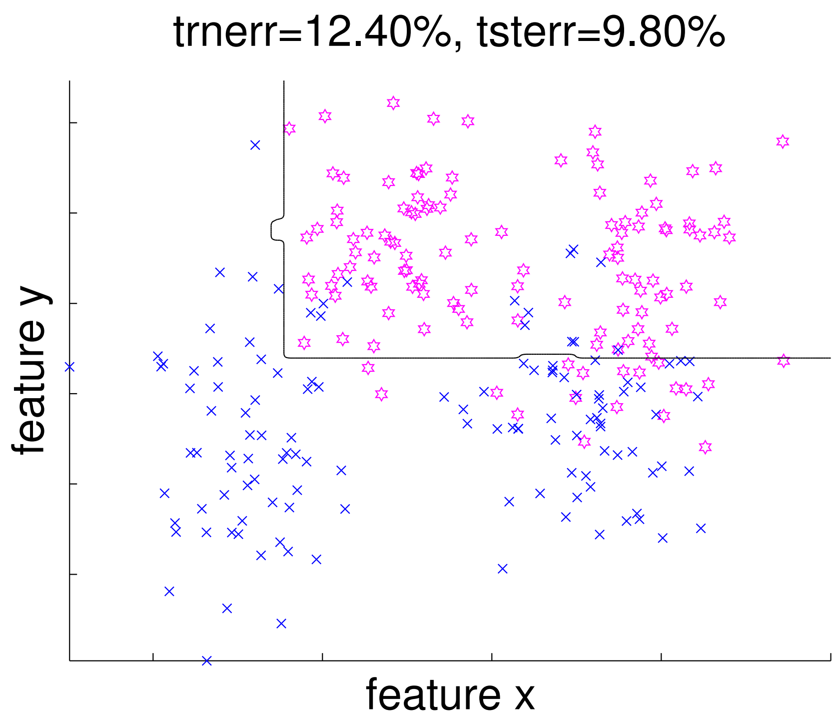


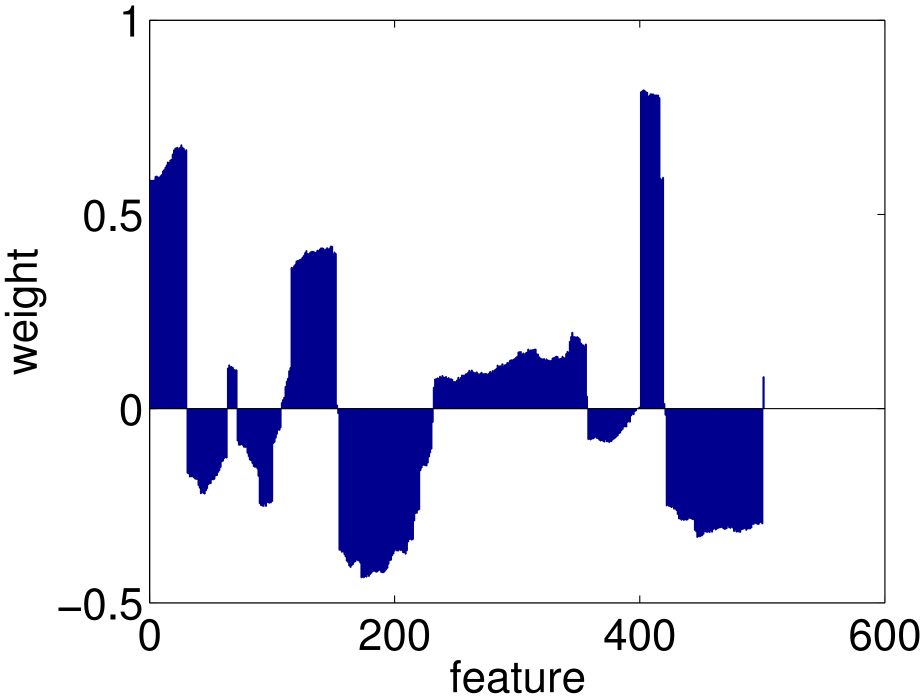
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



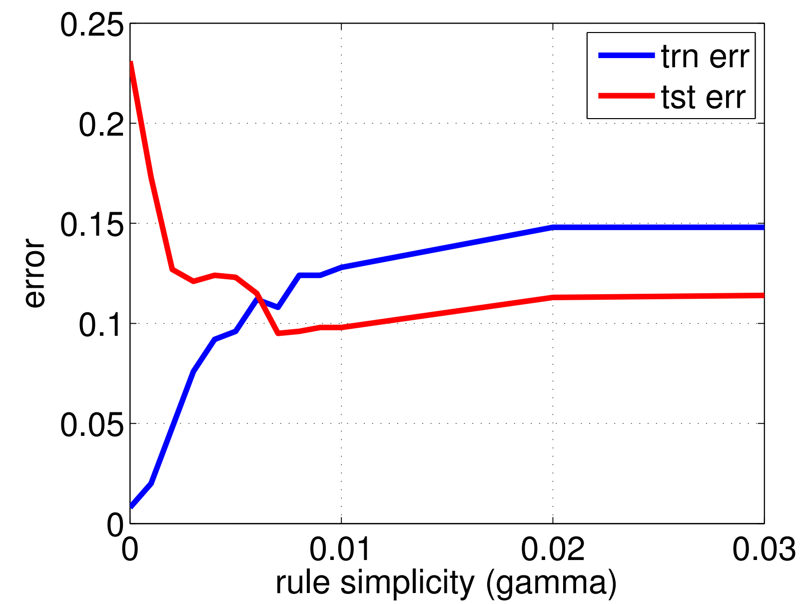
$\lambda = 0.001, \gamma = 0.009$



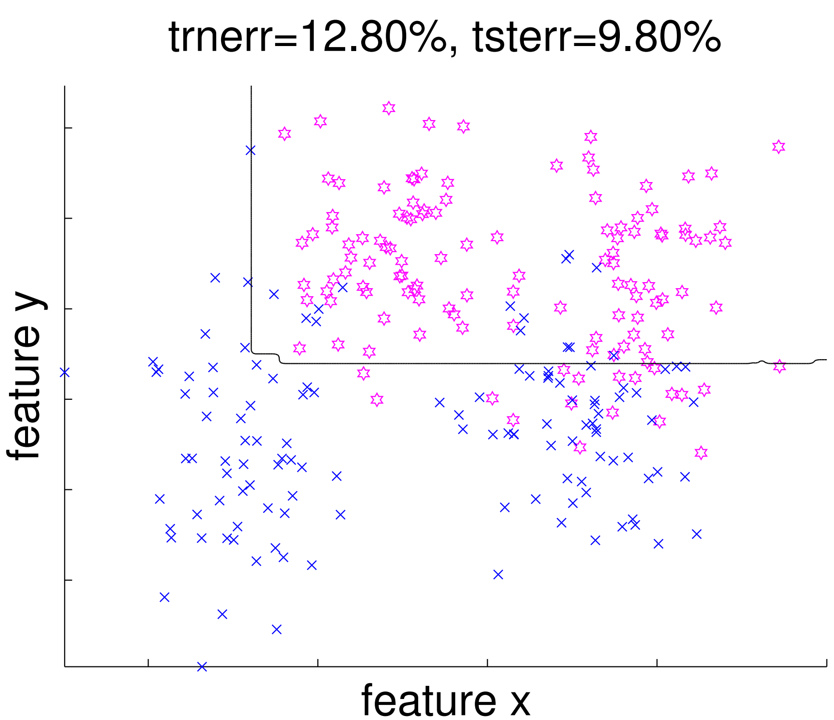


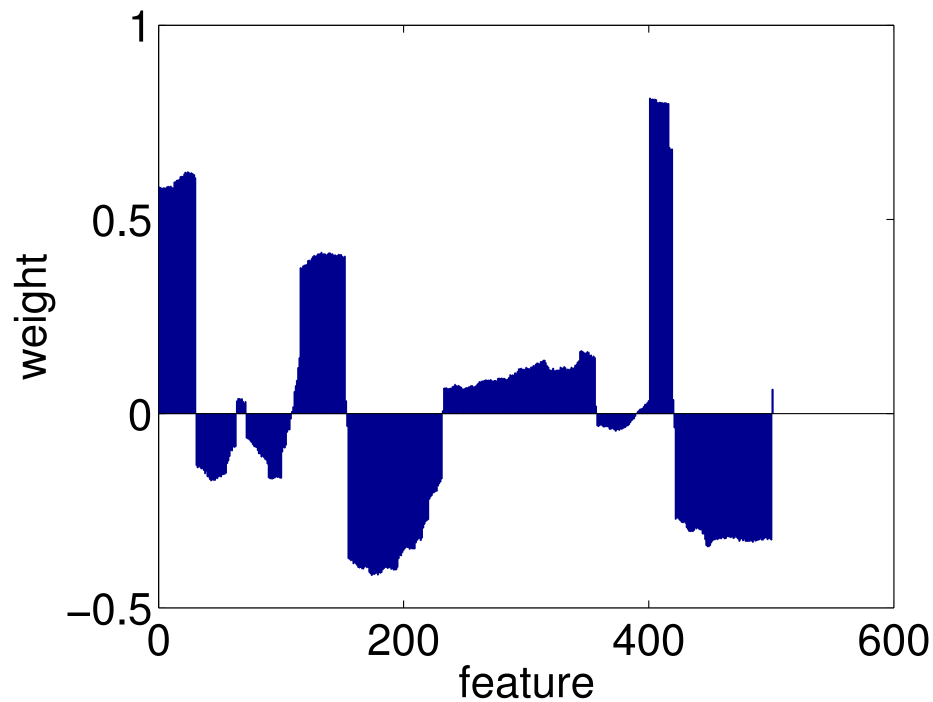
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



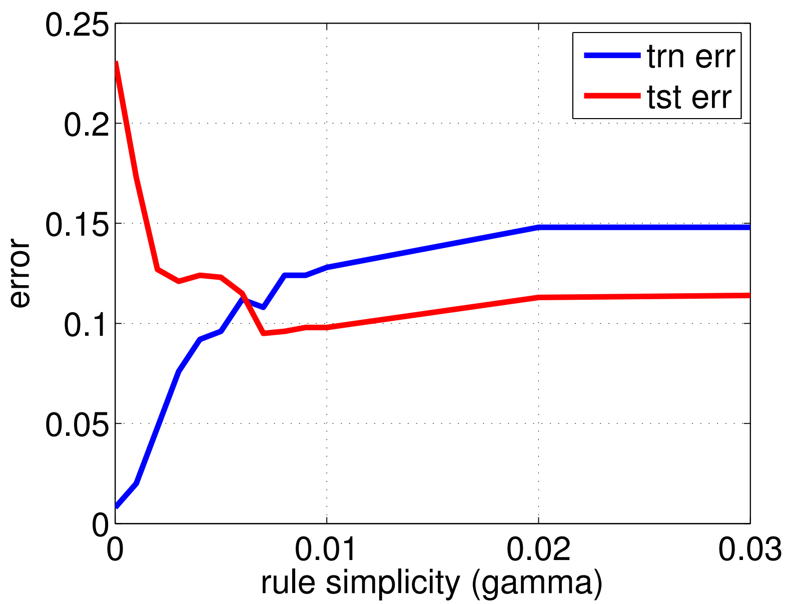
$\lambda = 0.001, \gamma = 0.010$



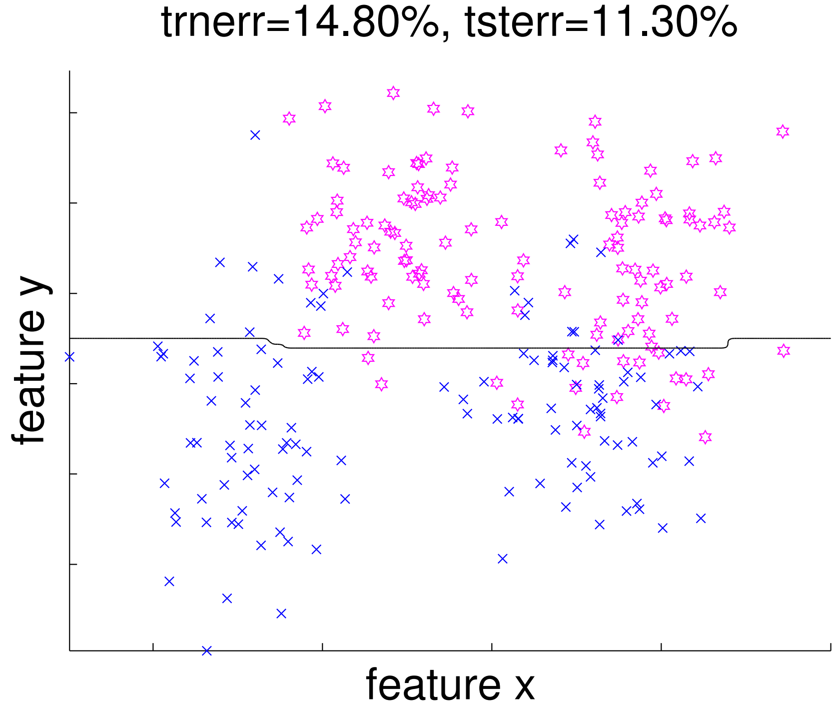


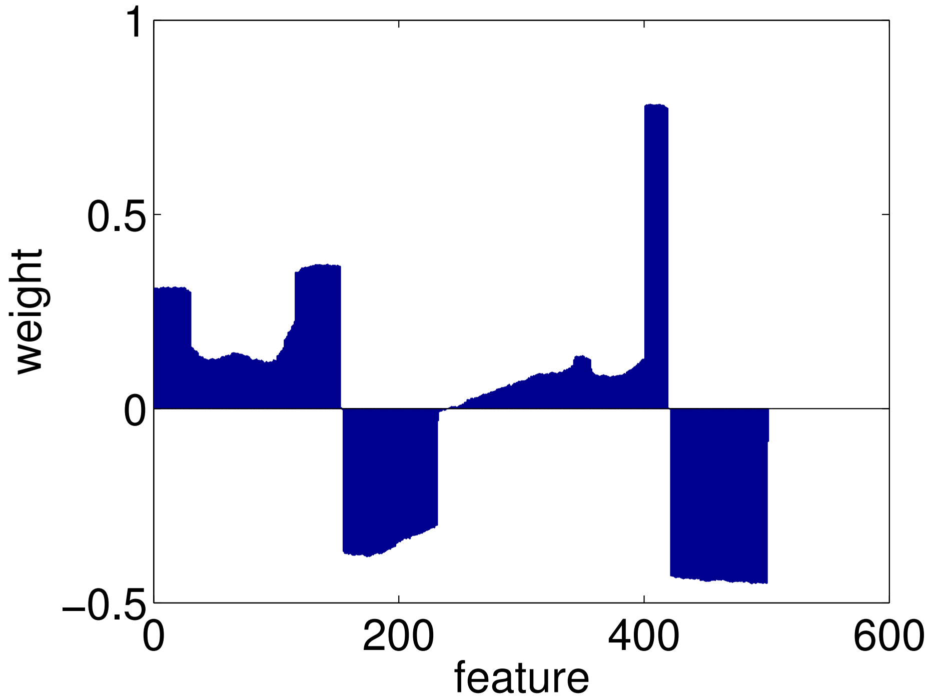
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



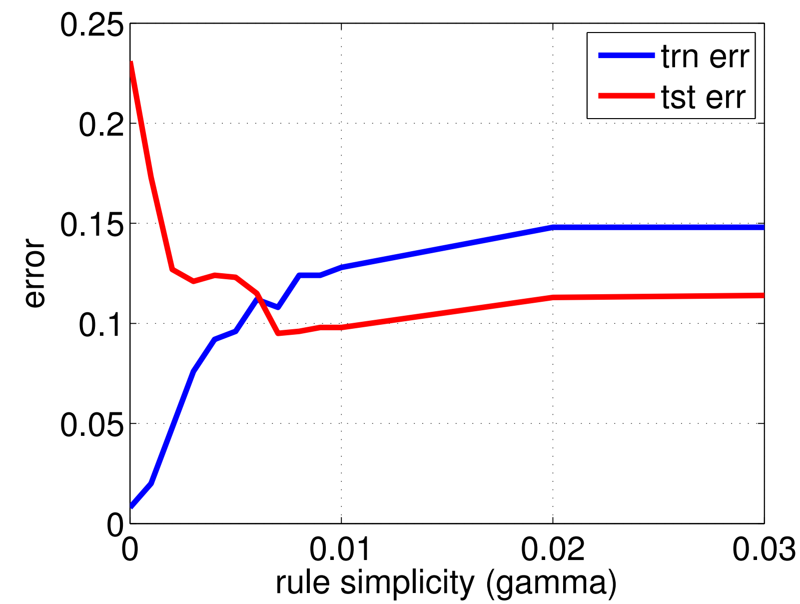
$$\lambda = 0.001, \ \gamma = 0.020$$



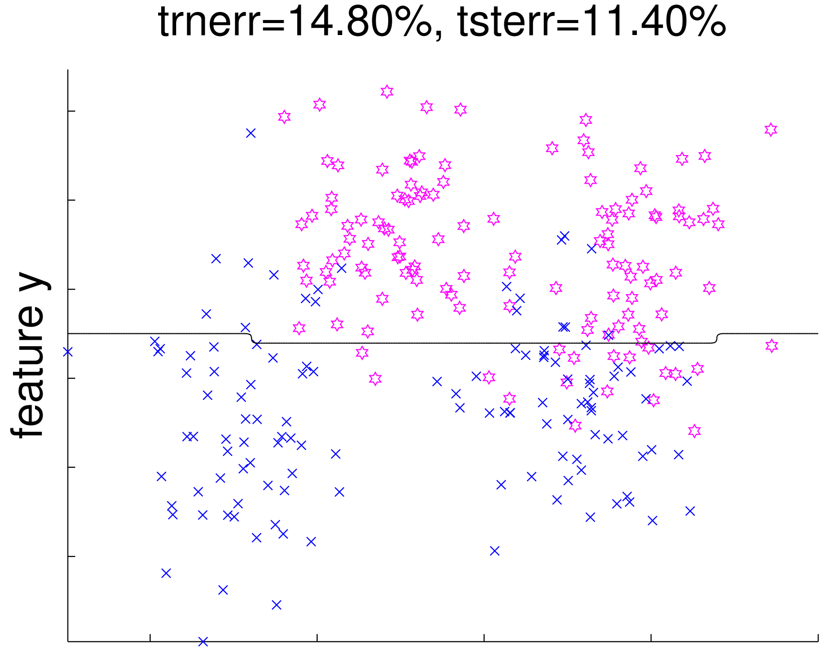


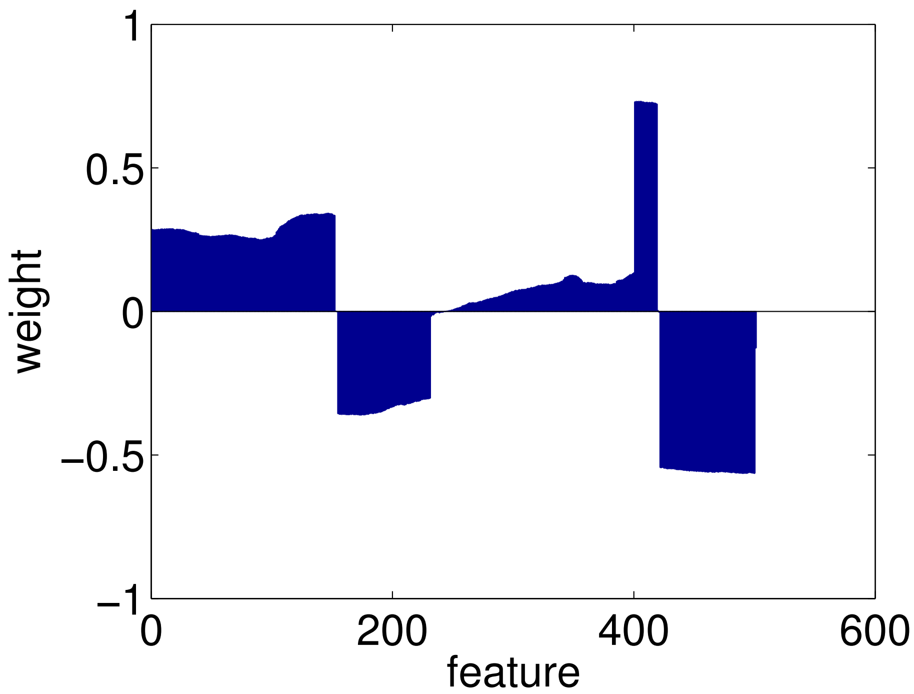
Example: Results on toy data

- A 2D point (x, y) is described by 5 real-valued features (x, y, x^2, y^2, xy).
- Each feature is discretized to $D = 100$ bins leading to $5 \cdot 100 = 500$ binary features.



$$\lambda = 0.001, \; \gamma = 0.030$$



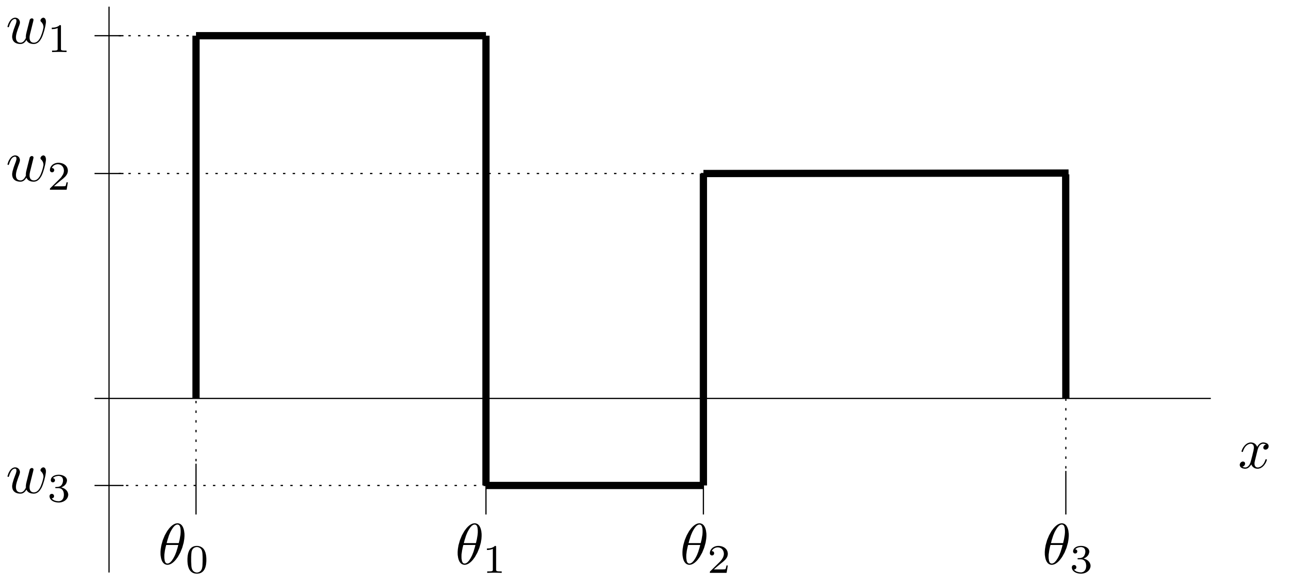


Learning piece-wise constant functions

We want to learn a piece-wise constant (PWC) function

$$f_{\text{pwc}}(x;\boldsymbol{w},\boldsymbol{\theta}) = \sum_{i=1}^{B} [\![x \in [\theta_{i-1},\theta_i)]\!] w_i = w_{k(x,\boldsymbol{\theta})}$$

where $x \in \mathbb{R}$ is the input variable, B is the number of bins, $\boldsymbol{\theta} = (\theta_0,\ldots,\theta_B)^T \in \mathbb{R}^{B+1}$ is the discretization of input variable and $\boldsymbol{w} = (w_1,\ldots,w_B)^T \in \mathbb{R}^B$ are the weights.



$B = 3$, $\boldsymbol{\theta} = (\theta_0,\ldots,\theta_3)^T$, $\boldsymbol{w} = (w_1,\ldots,w_3)^T$

Learning complete parametrization of PWC functions

Standard approach: for a fixed discretization $\boldsymbol{\theta}$ learn weights by a convex algorithm

$$\boldsymbol{w}^* \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^B} F_{\rm pwc}(\boldsymbol{w}, \boldsymbol{\theta}) := g\Big(f_{\rm pwc}(x^1; \boldsymbol{w}, \boldsymbol{\theta}), \ldots, f_{\rm pwc}(x^m; \boldsymbol{w}, \boldsymbol{\theta})\Big)$$

where $g \colon \mathbb{R}^m \to \mathbb{R}$ depends on $f_{\rm pwc}$ evaluated on a sample $\mathcal{T} = \{x^1, \ldots, x^m\} \in \mathbb{R}^m$.

Learning complete parametrization of PWC functions

Standard approach: for a fixed discretization $\boldsymbol{\theta}$ learn weights by a convex algorithm

$$\boldsymbol{w}^* \in \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^B} F_{\text{pwc}}(\boldsymbol{w},\boldsymbol{\theta}) := g\Big(f_{\text{pwc}}(x^1;\boldsymbol{w},\boldsymbol{\theta}),\ldots,f_{\text{pwc}}(x^m;\boldsymbol{w},\boldsymbol{\theta})\Big)$$

where $g\colon \mathbb{R}^m \to \mathbb{R}$ depends on f_{pwc} evaluated on a sample $\mathcal{T} = \{x^1,\ldots,x^m\} \in \mathbb{R}^m$.

We want to solve: learning the discretization and weights simultaneously

$$(\boldsymbol{w}^*,\boldsymbol{\theta}^*) \in \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^B,\boldsymbol{\theta}\in\Theta_B} F_{\text{pwc}}(\boldsymbol{w},\boldsymbol{\theta})$$

Learning complete parametrization of PWC functions

Standard approach: for a fixed discretization $\boldsymbol{\theta}$ learn weights by a convex algorithm

$$\boldsymbol{w}^* \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^B} F_{\text{pwc}}(\boldsymbol{w}, \boldsymbol{\theta}) := g\Big(f_{\text{pwc}}(x^1; \boldsymbol{w}, \boldsymbol{\theta}), \ldots, f_{\text{pwc}}(x^m; \boldsymbol{w}, \boldsymbol{\theta}) \Big)$$

where $g \colon \mathbb{R}^m \to \mathbb{R}$ depends on f_{pwc} evaluated on a sample $\mathcal{T} = \{x^1, \ldots, x^m\} \in \mathbb{R}^m$.

We want to solve: learning the discretization and weights simultaneously

$$(\boldsymbol{w}^*, \boldsymbol{\theta}^*) \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^B, \boldsymbol{\theta} \in \Theta_B} F_{\text{pwc}}(\boldsymbol{w}, \boldsymbol{\theta})$$

A set of admissible discretizations $\Theta_B \subset \mathbb{R}^{B+1}$ of the variable $x \in \mathbb{R}$ into B bins contains all vectors $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_B)^T$ satisfying:

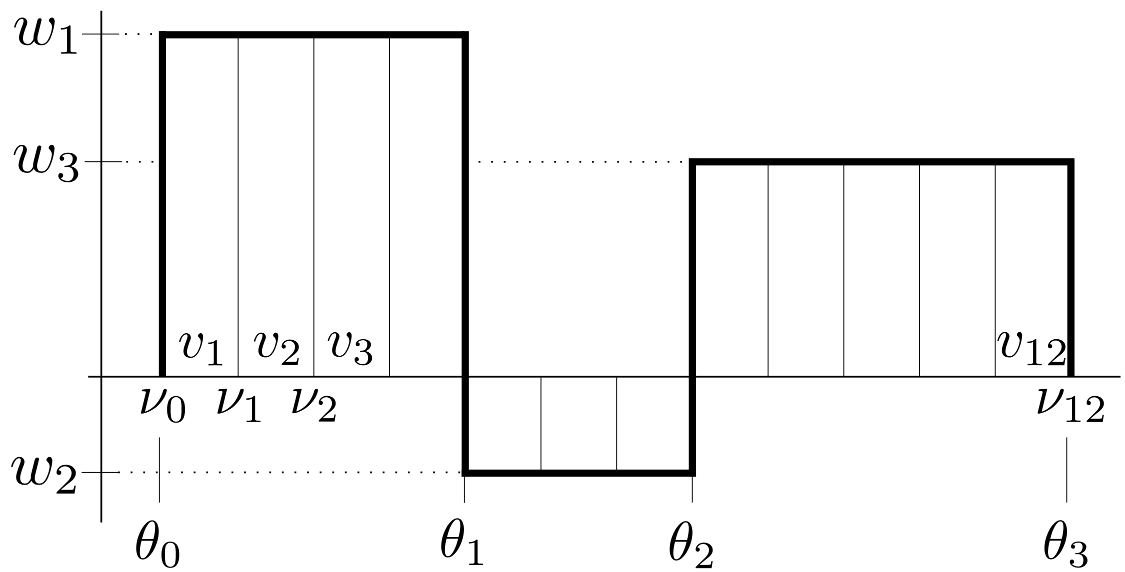
$$\theta_i = \nu_{l_i}\ , i \in \{0, \ldots, B\}, \qquad \text{where} \qquad \begin{array}{l} l_0 = 0\ , \\ l_i < l_{i+1}\ , \quad i \in \{1, \ldots, B-1\}\ , \\ l_B = D\ , \end{array}$$

and $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T \in \mathbb{R}^{D+1}$ is an initial discretization such that $\nu_0 < \cdots < \nu_D$.

Re-parametrization of PWC functions

For any $(\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^B \times \Theta_B)$ there exists a unique $\boldsymbol{v} \in \mathcal{V}_B = \{\boldsymbol{v} \in \mathbb{R}^D \,|\, c(\boldsymbol{v}) \leq B-1\}$, where $c(\boldsymbol{v}) = \sum_{i=1}^{D-1} [\![v_i \neq v_{i+1}]\!]$, such that

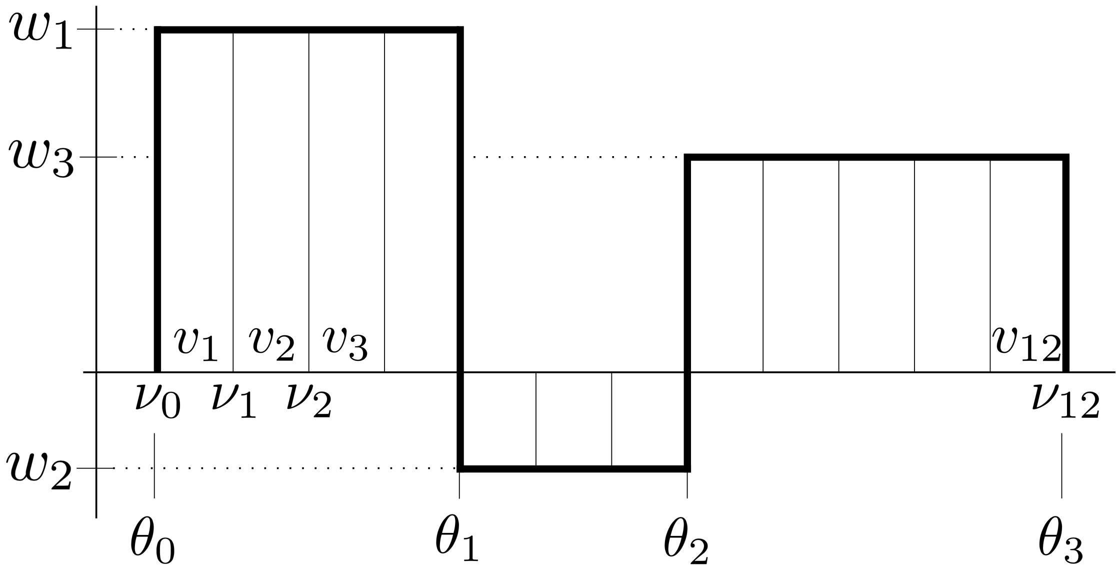
$$f_{\text{pwc}}(x; \boldsymbol{w}, \boldsymbol{\theta}) = f_{\text{pwc}}(x; \boldsymbol{v}, \boldsymbol{\nu})\,, \quad \forall x \in \mathbb{R}\,.$$



Re-parametrization of PWC functions

For any $(\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^B \times \Theta_B)$ there exists a unique $\boldsymbol{v} \in \mathcal{V}_B = \left\{\boldsymbol{v} \in \mathbb{R}^D \middle| c(\boldsymbol{v}) \leq B-1\right\}$, where $c(\boldsymbol{v}) = \sum_{i=1}^{D-1} [\![v_i \neq v_{i+1}]\!]$, such that

$$f_{\text{pwc}}(x; \boldsymbol{w}, \boldsymbol{\theta}) = f_{\text{pwc}}(x; \boldsymbol{v}, \boldsymbol{\nu})\,, \quad \forall x \in \mathbb{R}\,.$$



The equivalence between the two parametrizations implies that

$$\min\left\{F_{\text{pwc}}(\boldsymbol{w}; \boldsymbol{\theta}) \mid (\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^B \times \Theta_B)\right\} = \min\left\{F_{\text{pwc}}(\boldsymbol{v}; \boldsymbol{\nu}) \mid \boldsymbol{v} \in \mathcal{V}_B\right\}$$

Learning PWC functions via convex programming

The original problem: Learning of the discretization and the weights of a PWC function is equivalent to solving

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^D} F_{\text{pwc}}(\boldsymbol{v},\boldsymbol{\nu}) \quad \text{s.t.} \quad c(\boldsymbol{v}) \leq B-1\,, \tag{*}$$

and using $\boldsymbol{v}^*$ to recover the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$.

Learning PWC functions via convex programming

The original problem: Learning of the discretization and the weights of a PWC function is equivalent to solving

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^D} F_{\text{pwc}}(\boldsymbol{v},\boldsymbol{\nu}) \quad \text{s.t.} \quad c(\boldsymbol{v}) \leq B-1\,, \tag{*}$$

and using $\boldsymbol{v}^*$ to recover the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$.

A convex relaxation of the problem (*) reads

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^D} F_{\text{pwc}}(\boldsymbol{v};\boldsymbol{\nu}) \quad \text{s.t.} \quad \sum_{i=1}^{D-1} |v_i - v_{i+1}| \leq B-1\,,$$

where $c(\boldsymbol{v}) = \sum_{i=1}^{D-1} [\![v_i \neq v_{i+1}]\!] = \|\boldsymbol{d}\|_0$, $\boldsymbol{d} = (v_1 - v_2, \dots, v_{D-1} - v_D)^T$, is replaced by the L_1-norm $\tilde{c}(\boldsymbol{v}) = \|\boldsymbol{d}\|_1$.

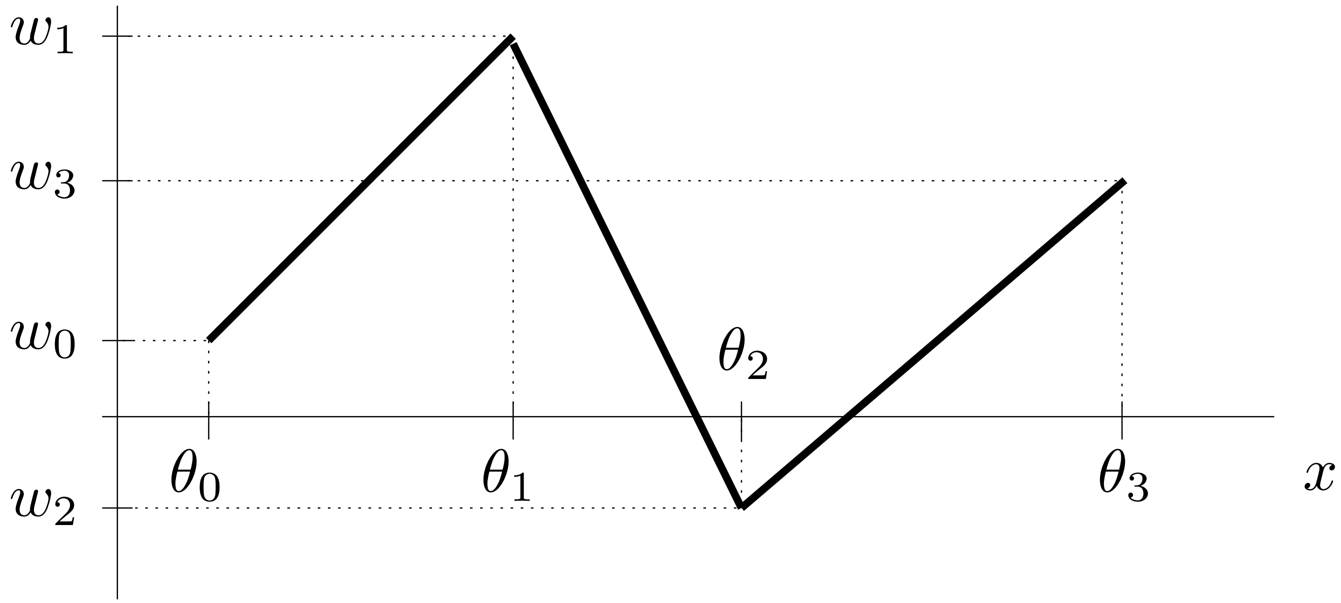
Learning piece-wise linear functions

We want to learn a piece-wise linear (PWL) function

$$f_{\text{pwl}}(x; \boldsymbol{w}, \boldsymbol{\theta}) = w_{k(x,\boldsymbol{\theta})-1} \cdot (1 - \alpha(x, \boldsymbol{\theta})) + w_{k(x,\boldsymbol{\theta})} \cdot \alpha(x, \boldsymbol{\theta}))$$

where $x \in \mathbb{R}$ is the input variable, $\boldsymbol{\theta} \in \mathbb{R}^{B+1}$ is the discretization, B is the number of bins, $\boldsymbol{w} \in \mathbb{R}^{B+1}$ are the weights and $\alpha \colon \mathbb{R} \times \mathbb{R}^{B+1} \to [0, 1]$ is defined as

$$\alpha(x, \boldsymbol{\theta}) = \frac{x - \theta_{k(x,\boldsymbol{\theta})-1}}{\theta_{k(x,\boldsymbol{\theta})} - \theta_{k(x,\boldsymbol{\theta})-1}}$$



$B = 3$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_3)^T$, $\boldsymbol{w} = (w_0, w_1, \ldots, w_3)^T$

Learning piece-wise linear functions

Task formulation: We want to learn the discretization $\boldsymbol{\theta}^* \in \Theta_B$ simultaneously with the weights $\boldsymbol{w}^* \in \mathbb{R}^B$ by solving

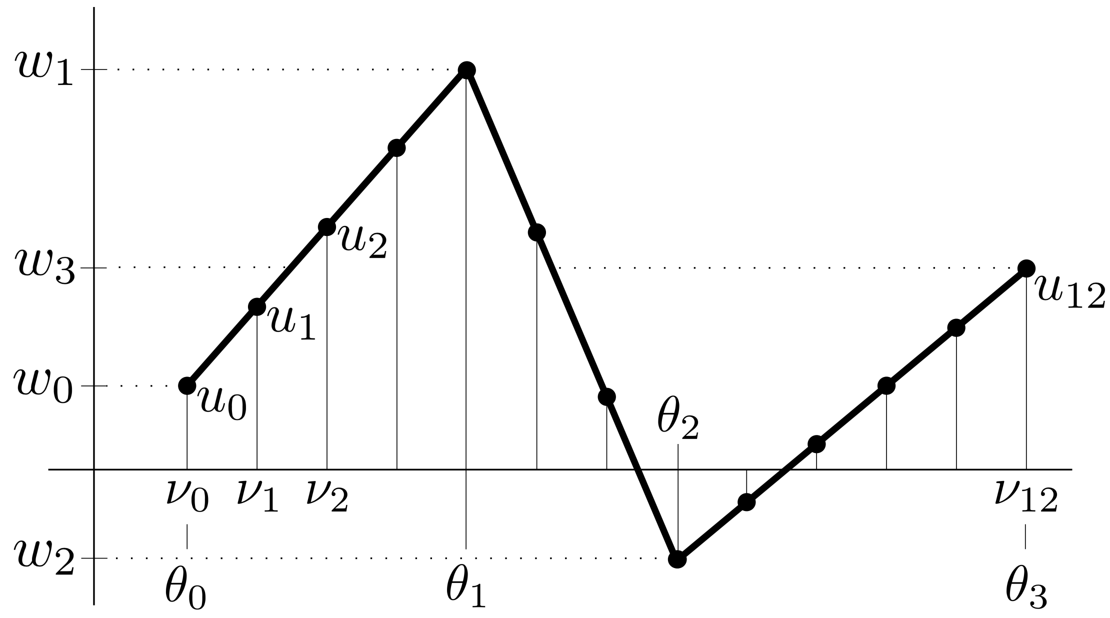
$$(\boldsymbol{w}^*, \boldsymbol{\theta}^*) \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^{B+1}, \boldsymbol{\theta} \in \Theta_B} F_{\text{pwl}}(\boldsymbol{w}, \boldsymbol{\theta}) := g\Big(f_{\text{pwl}}(x^1; \boldsymbol{w}, \boldsymbol{\theta}), \ldots, f_{\text{pwl}}(x^m; \boldsymbol{w}, \boldsymbol{\theta})\Big)$$

where $g \colon \mathbb{R}^m \to \mathbb{R}$ depends on f_{pwl} evaluated on a sample $\mathcal{T} = \{x^1, \ldots, x^m\} \in \mathbb{R}^m$.

Re-parametrization of PWL functions

For any $(\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^{B+1} \times \Theta_B)$ there exists uniq $\boldsymbol{u} \in \mathcal{U}_B = \{\boldsymbol{u} \in \mathbb{R}^{D+1} | e(\boldsymbol{u}) \leq B-1\}$, where $e(\boldsymbol{u}) = \sum_{i=1}^{D-1} [\![u_i \neq \frac{1}{2}(u_{i-1} + u_{i+1})]\!]$, such that

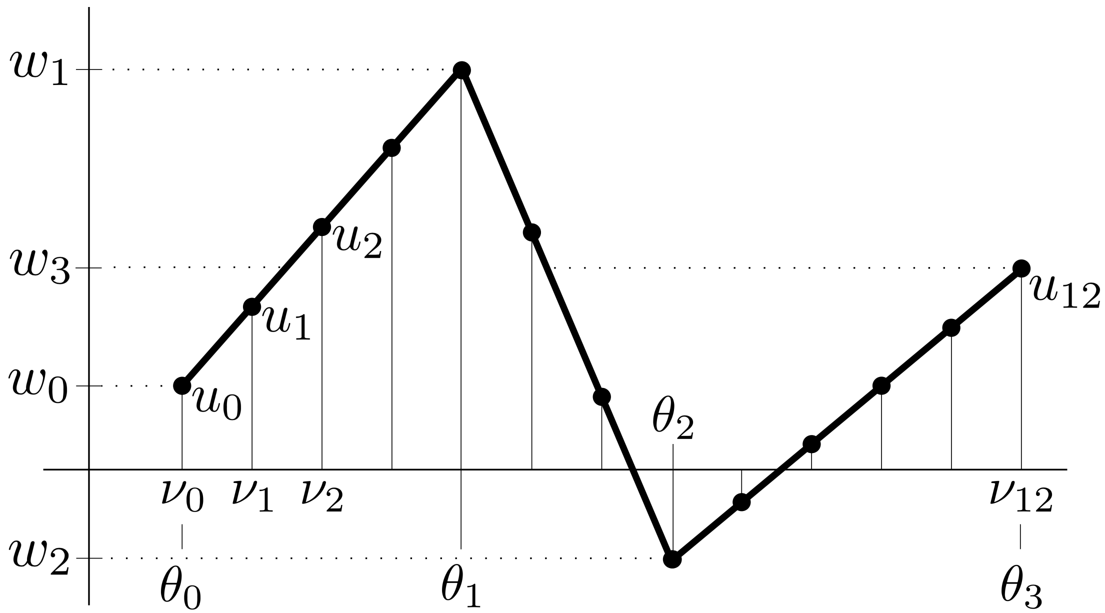
$$f_{\text{pwl}}(x; \boldsymbol{w}, \boldsymbol{\theta}) = f_{\text{pwl}}(x; \boldsymbol{u}, \boldsymbol{\nu}), \quad \forall \boldsymbol{x} \in \mathbb{R}.$$



Re-parametrization of PWL functions

For any $(\boldsymbol{w},\boldsymbol{\theta}) \in (\mathbb{R}^{B+1} \times \Theta_B)$ there exists uniq $\boldsymbol{u} \in \mathcal{U}_B = \{\boldsymbol{u} \in \mathbb{R}^{D+1} \,|\, e(\boldsymbol{u}) \leq B-1\}$, where $e(\boldsymbol{u}) = \sum_{i=1}^{D-1} [\![u_i \neq \frac{1}{2}(u_{i-1}+u_{i+1})]\!]$, such that

$$f_{\text{pwl}}(x;\boldsymbol{w},\boldsymbol{\theta}) = f_{\text{pwl}}(x;\boldsymbol{u},\boldsymbol{\nu})\,, \quad \forall \boldsymbol{x} \in \mathbb{R}\,.$$



The equivalence between the two parametrizations implies that

$$\min\left\{F_{\text{pwl}}(\boldsymbol{w};\boldsymbol{\theta}) \mid (\boldsymbol{w},\boldsymbol{\theta}) \in (\mathbb{R}^{B+1} \times \Theta_B)\right\} = \min\left\{F_{\text{pwl}}(\boldsymbol{u};\boldsymbol{\nu}) \mid \boldsymbol{u} \in \mathcal{U}_B\right\}$$

Learning PWL functions via convex programming

The original problem: Learning of the discretization and the weights of PWL function is equivalent to solving

$$\boldsymbol{u}^* \in \operatorname*{argmin}_{\boldsymbol{u}\in\mathbb{R}^{D+1}} F_{\text{pwl}}(\boldsymbol{u}, \boldsymbol{\nu}) \quad \text{s.t.} \quad e(\boldsymbol{u}) \leq B - 1\,, \tag{*}$$

and using $\boldsymbol{u}^*$ to recover the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$.

Learning PWL functions via convex programming

The original problem: Learning of the discretization and the weights of PWL function is equivalent to solving

$$\boldsymbol{u}^* \in \operatorname*{argmin}_{\boldsymbol{u}\in\mathbb{R}^{D+1}} F_{\text{pwl}}(\boldsymbol{u},\boldsymbol{\nu}) \quad \text{s.t.} \quad e(\boldsymbol{u}) \le B-1\,, \qquad (*)$$

and using $\boldsymbol{u}^*$ to recover the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$.

A convex relaxation: of the problem (*) reads

$$\boldsymbol{u}^* \in \operatorname*{argmin}_{\boldsymbol{u}\in\mathbb{R}^{D+1}} F_{\text{pwl}}(\boldsymbol{u},\boldsymbol{\nu}) \quad \text{s.t} \quad \sum_{i=1}^{D-1} \left| u_i - \frac{1}{2}u_{i-1} - \frac{1}{2}u_{i+1} \right| \le B-1\,,$$

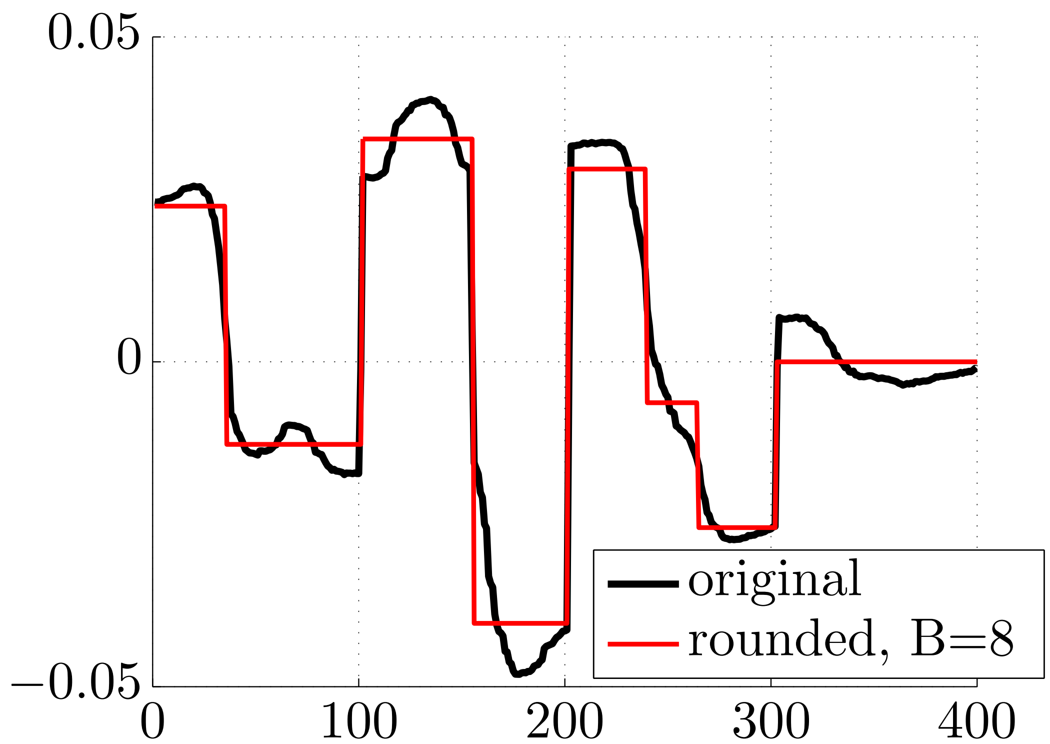
where $e(\boldsymbol{u}) = \sum_{i=1}^{D-1} [\![u_i \neq \frac{1}{2}(u_{i-1}+u_{i+1})]\!] = \|\boldsymbol{d}\|_0$,
$\boldsymbol{d} = (u_1 - \frac{1}{2}(u_0+u_2), \ldots, u_D - \frac{1}{2}(u_{D-1}+u_{D+1}))^T$, is replaced by the L_1-norm proxy.

Recovering the bins by rounding the solution

- The histograms emerging in the parameters $\boldsymbol{v} \in \mathbb{R}^D$ are not perfect ($c(\boldsymbol{v})$ is often high) due to the L_1-norm approximation and numerical errors.
- For given $\boldsymbol{v}$, a rounded solution $\bar{\boldsymbol{v}}$ with B bins can be found by

$$\bar{\boldsymbol{v}} \in \operatorname*{argmin}_{\boldsymbol{v}' \in \mathbb{R}^D} \|\boldsymbol{v} - \boldsymbol{v}'\|^2 \quad \text{s.t.} \quad c(\boldsymbol{v}') = B - 1$$

which can be solver in $\mathcal{O}(D^2 \cdot B)$ time by dynamic programming.



Example application: classification of histograms

The classification model: a linear classifier $h(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \mathrm{sign}(f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}))$ assigning $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, which describes n sequences of d elements, according to sign of

$$f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \sum_{i=1}^{n}\sum_{j=1}^{b_i}\frac{1}{d}\sum_{k=1}^{d}[\![X_{i,k} \in [\theta_{i,j-1},\theta_{i,j})]\!]w_{i,j}$$

Example application: classification of histograms

The classification model: a linear classifier $h(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \mathrm{sign}(f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}))$ assigning $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, which describes n sequences of d elements, according to sign of

$$f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \sum_{i=1}^{n}\sum_{j=1}^{b_i}\frac{1}{d}\sum_{k=1}^{d}[\![X_{i,k} \in [\theta_{i,j-1},\theta_{i,j})]\!]w_{i,j}$$

The task: learn $\boldsymbol{w} \in \mathbb{R}^B$ and $\boldsymbol{\theta} \in \Theta_B$, where Θ_B is induced by $\nu \in \mathbb{R}^{D\cdot n}$, by solving

$$(\boldsymbol{w}^*,\boldsymbol{\theta}^*) = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^B,\boldsymbol{\theta}\in\Theta_B} F^{\mathrm{svm}}_{\mathrm{pwc}}(\boldsymbol{w},\boldsymbol{\theta};\lambda) := \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\max\left\{0, 1 - y_i f_{\mathrm{pwc}}(\boldsymbol{x}^i,\boldsymbol{w},\boldsymbol{\theta})\right\}$$

Example application: classification of histograms

The classification model: a linear classifier $h(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \mathrm{sign}(f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}))$ assigning $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, which describes n sequences of d elements, according to sign of

$$f_{\mathrm{pwc}}(\boldsymbol{X};\boldsymbol{w},\boldsymbol{\theta}) = \sum_{i=1}^{n}\sum_{j=1}^{b_i}\frac{1}{d}\sum_{k=1}^{d}[\![X_{i,k} \in [\theta_{i,j-1},\theta_{i,j})]\!]w_{i,j}$$

The task: learn $\boldsymbol{w} \in \mathbb{R}^B$ and $\boldsymbol{\theta} \in \Theta_B$, where Θ_B is induced by $\nu \in \mathbb{R}^{D\cdot n}$, by solving

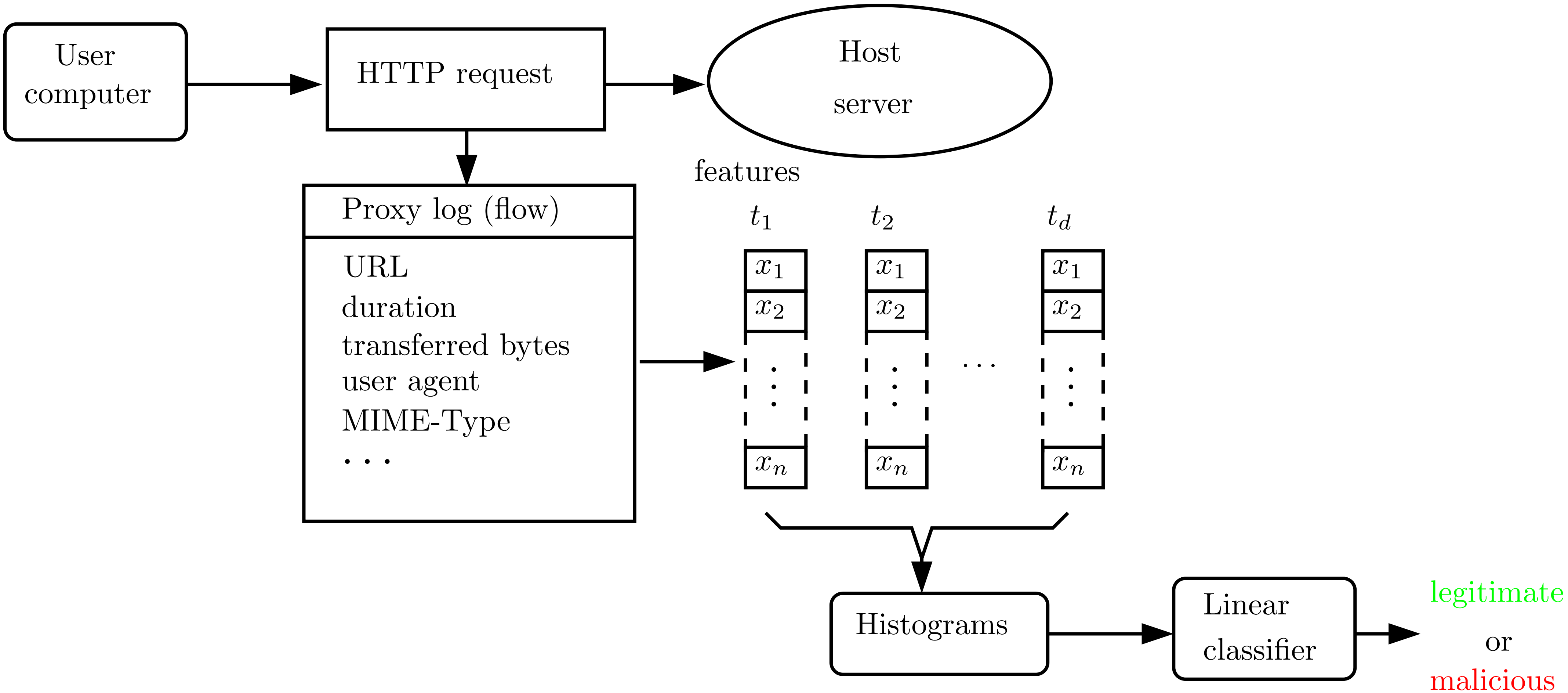
$$(\boldsymbol{w}^*,\boldsymbol{\theta}^*) = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^B,\boldsymbol{\theta}\in\Theta_B} F^{\mathrm{svm}}_{\mathrm{pwc}}(\boldsymbol{w},\boldsymbol{\theta};\lambda) := \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\max\left\{0, 1 - y_i f_{\mathrm{pwc}}(\boldsymbol{x}^i,\boldsymbol{w},\boldsymbol{\theta})\right\}$$

The convex relaxation: Learning is converted to a convex program

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^{nD}}\left[F^{\mathrm{svm}}_{\mathrm{pwc}}(\boldsymbol{v},\boldsymbol{\nu};\lambda) + \gamma\sum_{i=1}^{n}\sum_{j=1}^{D-1}|v_{i,j} - v_{i,j+1}|\right].$$

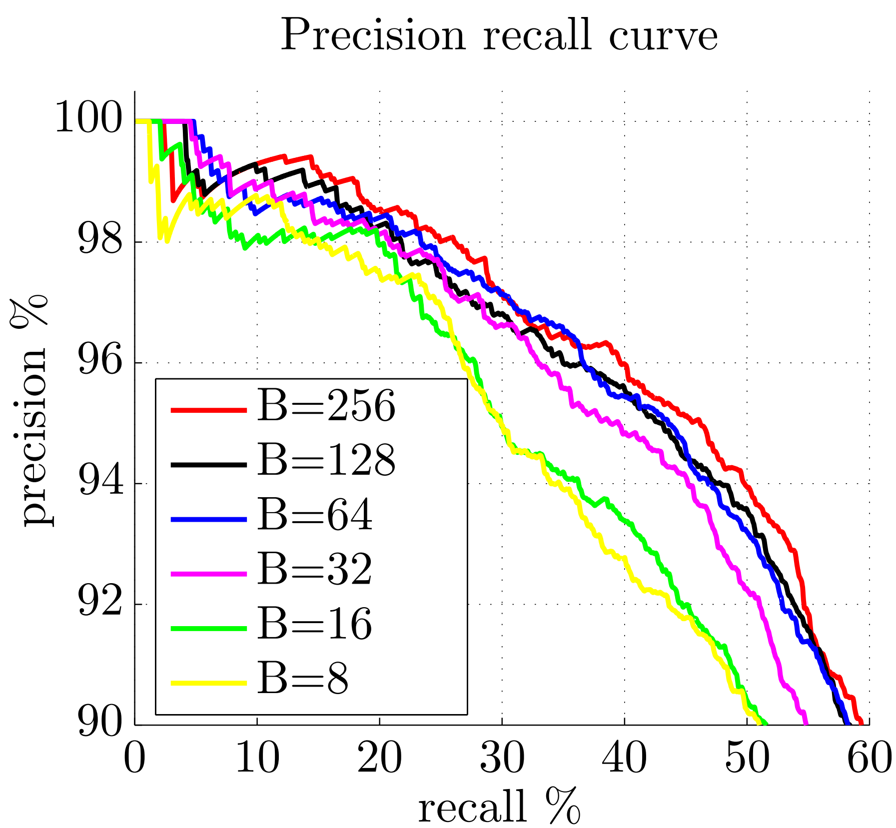
Experiments: malware detection

Task is to detect malware based on sequences of features which are extracted from HTTP proxy logs describing communication between a user computer and a server.

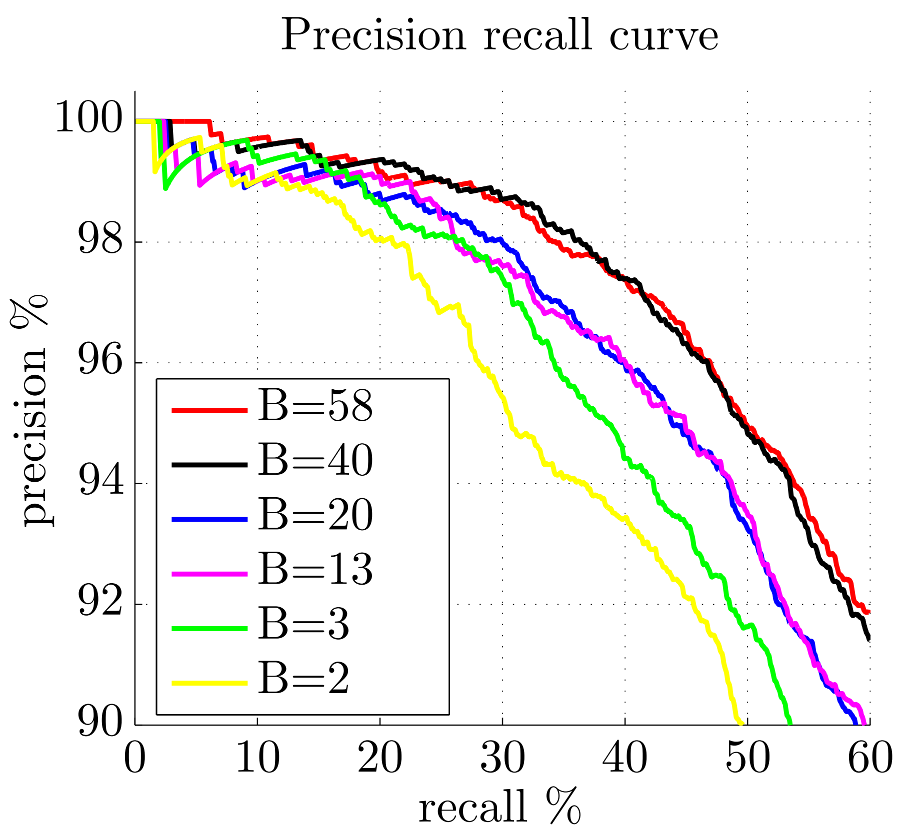


Training data has 7,028 positive (malware) and 44,338 negative (legitimate) samples.

Experiments: malware detection

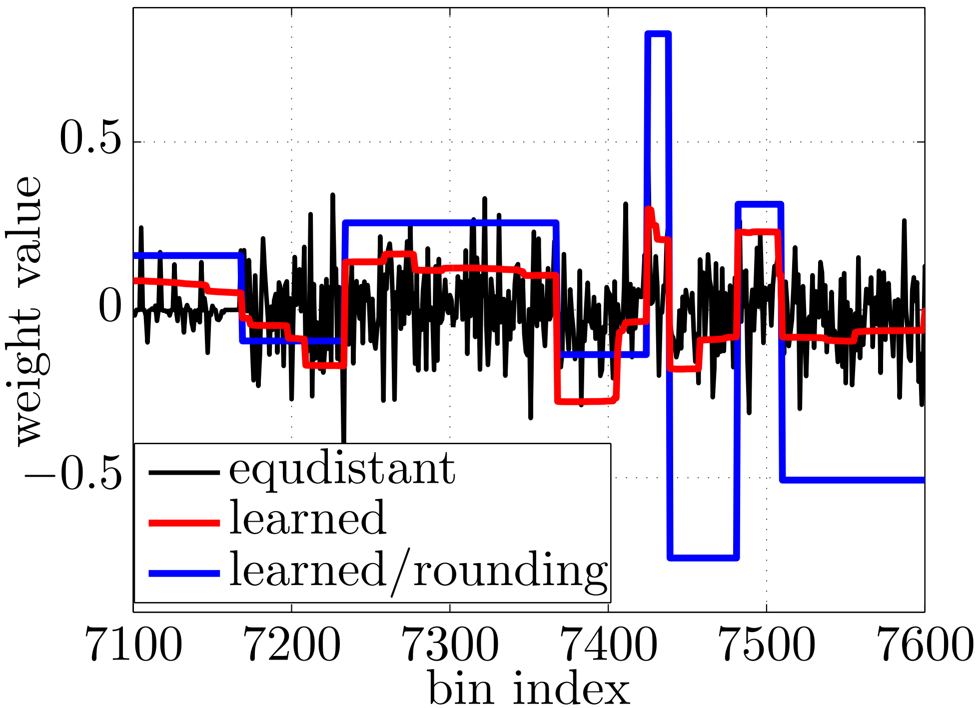


Precision recall curves for histogram representation with different number of **equidistant bins**.

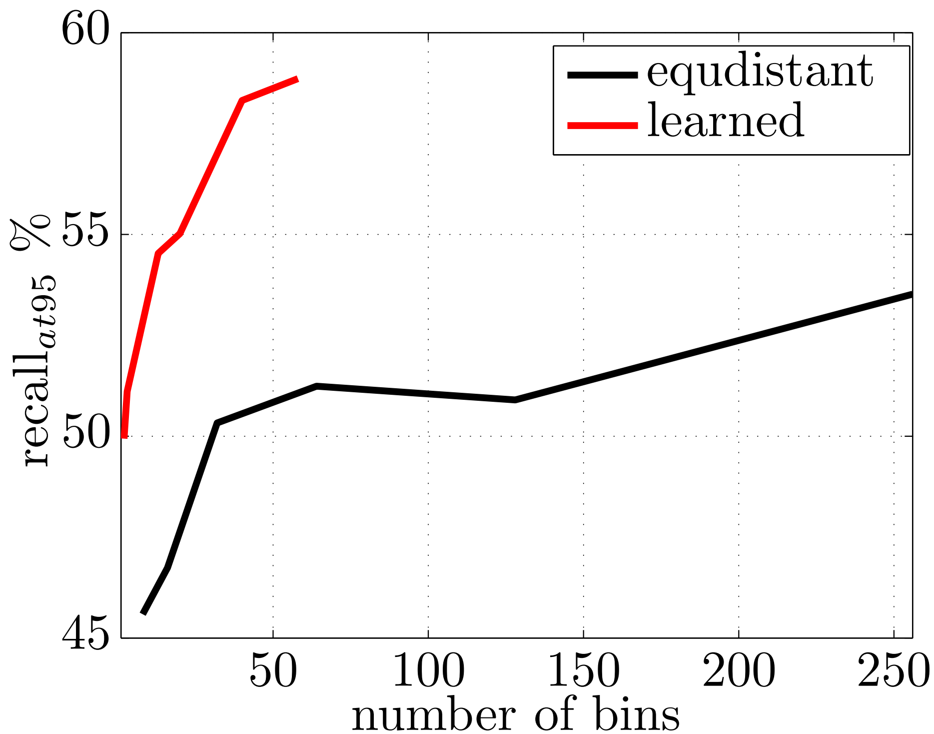


Precision recall curves for histogram representation with **non-equidistant bins learned** by the proposed method.

Experiments: malware detection



SVM weights learned with three different algorithms: a) linear SVM with 256 equidistant bins, b) proposed simultaneous learning of weights and bins, c) same as b) to define new bins (rounding) that are used for learning new linear SVM classifier.



Recall at precision 95% as a function of the number of bins for the representation with a) equidistant bins and b) learned non-equidistant bins.

Example application: learning PWL histograms

The model: probability density modeled by a PWL function

$$\hat{p}_{\mathrm{pwl}}(x;\boldsymbol{w},\boldsymbol{\theta}) = \big(1-\alpha(x,\boldsymbol{\theta})\big)w_{k(x,\boldsymbol{\theta})-1} + \alpha(x,\boldsymbol{\theta})w_{k(x,\theta)}$$

where $\boldsymbol{w} \in \mathbb{R}_+^{B+1}$ are non-negative weights selected such that $\int \hat{p}_{\mathrm{pwl}}(x;\boldsymbol{w},\boldsymbol{\theta})\mathrm{d}x = 1$.

Example application: learning PWL histograms

The model: probability density modeled by a PWL function

$$\hat{p}_{\mathrm{pwl}}(x;\boldsymbol{w},\boldsymbol{\theta}) = \big(1-\alpha(x,\boldsymbol{\theta})\big)w_{k(x,\boldsymbol{\theta})-1} + \alpha(x,\boldsymbol{\theta})w_{k(x,\theta)}$$

where $\boldsymbol{w}\in\mathbb{R}_+^{B+1}$ are non-negative weights selected such that $\int \hat{p}_{\mathrm{pwl}}(x;\boldsymbol{w},\boldsymbol{\theta})\mathrm{d}x = 1$.

Task: given $\mathcal{T}=\{x^1,\ldots,x^m\}\in\mathbb{R}^m$, learn the parameters $\boldsymbol{w}\in\mathcal{W}$ and $\boldsymbol{\theta}\in\Theta_B$ by

$$(\boldsymbol{w}^*,\boldsymbol{\theta}^*) = \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{W},\boldsymbol{\theta}\in\Theta_B} F^{\mathrm{nnl}}_{\mathrm{pwl}}(\boldsymbol{w},\boldsymbol{\theta}) := -\sum_{i=1}^{m}\log\hat{p}_{\mathrm{pwl}}(x^i;\boldsymbol{w},\boldsymbol{\theta})$$

Example application: learning PWL histograms

The model: probability density modeled by a PWL function

$$\hat{p}_{\rm pwl}(x;\boldsymbol{w},\boldsymbol{\theta}) = \big(1-\alpha(x,\boldsymbol{\theta})\big) w_{k(x,\boldsymbol{\theta})-1} + \alpha(x,\boldsymbol{\theta}) w_{k(x,\theta)}$$

where $\boldsymbol{w} \in \mathbb{R}_+^{B+1}$ are non-negative weights selected such that $\int \hat{p}_{\rm pwl}(x;\boldsymbol{w},\boldsymbol{\theta}){\rm d}x = 1$.

Task: given $\mathcal{T} = \{x^1,\ldots,x^m\} \in \mathbb{R}^m$, learn the parameters $\boldsymbol{w} \in \mathcal{W}$ and $\boldsymbol{\theta} \in \Theta_B$ by

$$(\boldsymbol{w}^*,\boldsymbol{\theta}^*) = \mathop{\rm argmin}_{\boldsymbol{w}\in\mathcal{W},\boldsymbol{\theta}\in\Theta_B} F_{\rm pwl}^{\rm nnl}(\boldsymbol{w},\boldsymbol{\theta}) := -\sum_{i=1}^{m} \log \hat{p}_{\rm pwl}(x^i;\boldsymbol{w},\boldsymbol{\theta})$$

The convex relaxation: Learning is converted to a convex program

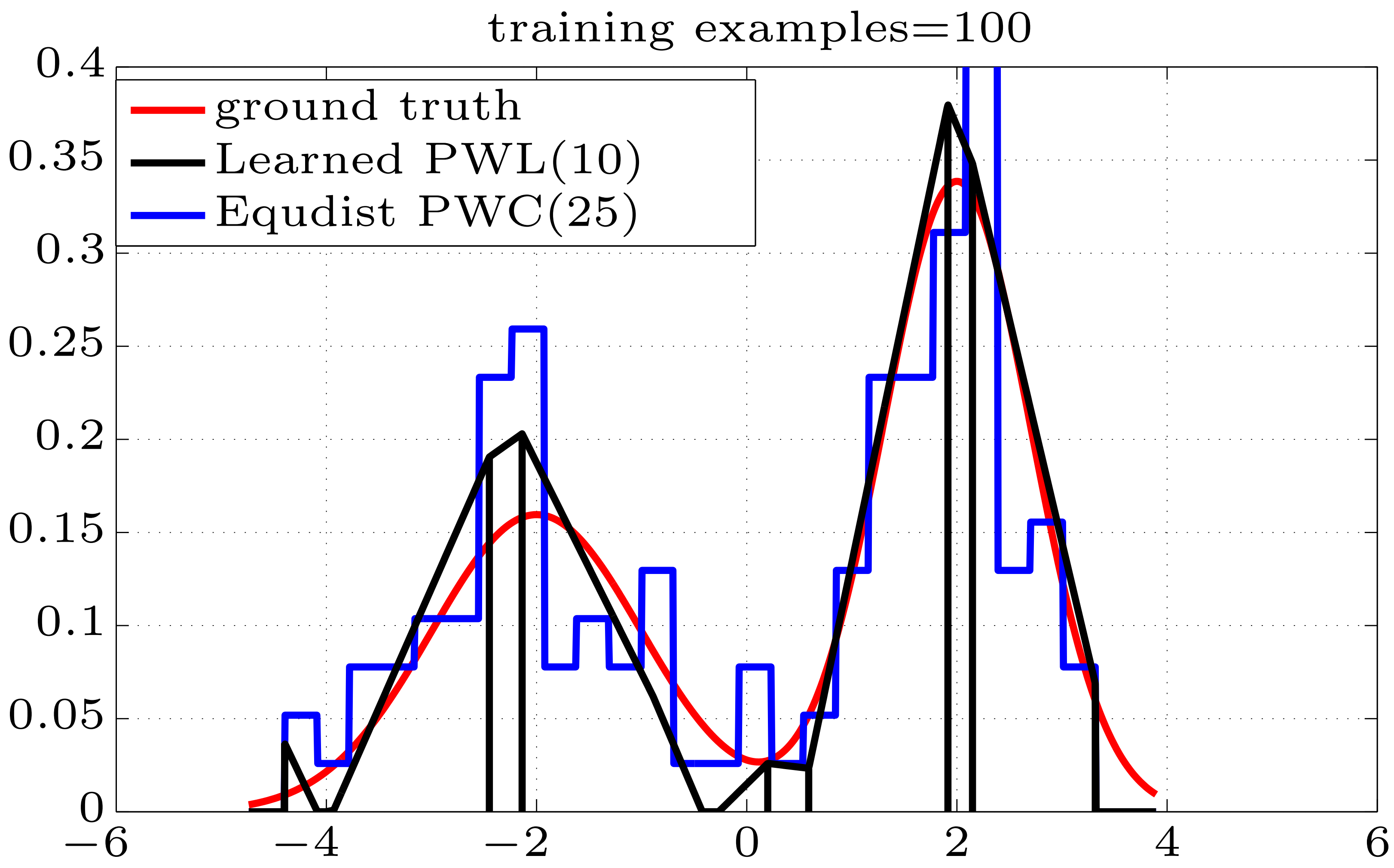
$$\boldsymbol{u}^* = \mathop{\rm argmin}_{\boldsymbol{u}\in\mathbb{R}^D} \left[F_{\rm pwl}^{\rm nnl}(\boldsymbol{u},\boldsymbol{\nu}) + \gamma \sum_{j=1}^{D-1} \left| u_j - \frac{1}{2}u_{j-1} - \frac{1}{2}u_{j+1} \right| \right]$$

subject to

$$u_0 + u_D + 2\sum_{i=1}^{D-1} u_i = \frac{2D}{\text{Max} - \text{Min}}\,, \qquad u_i \geq 0\,, \quad i \in \{0,\ldots,D\}\,,$$

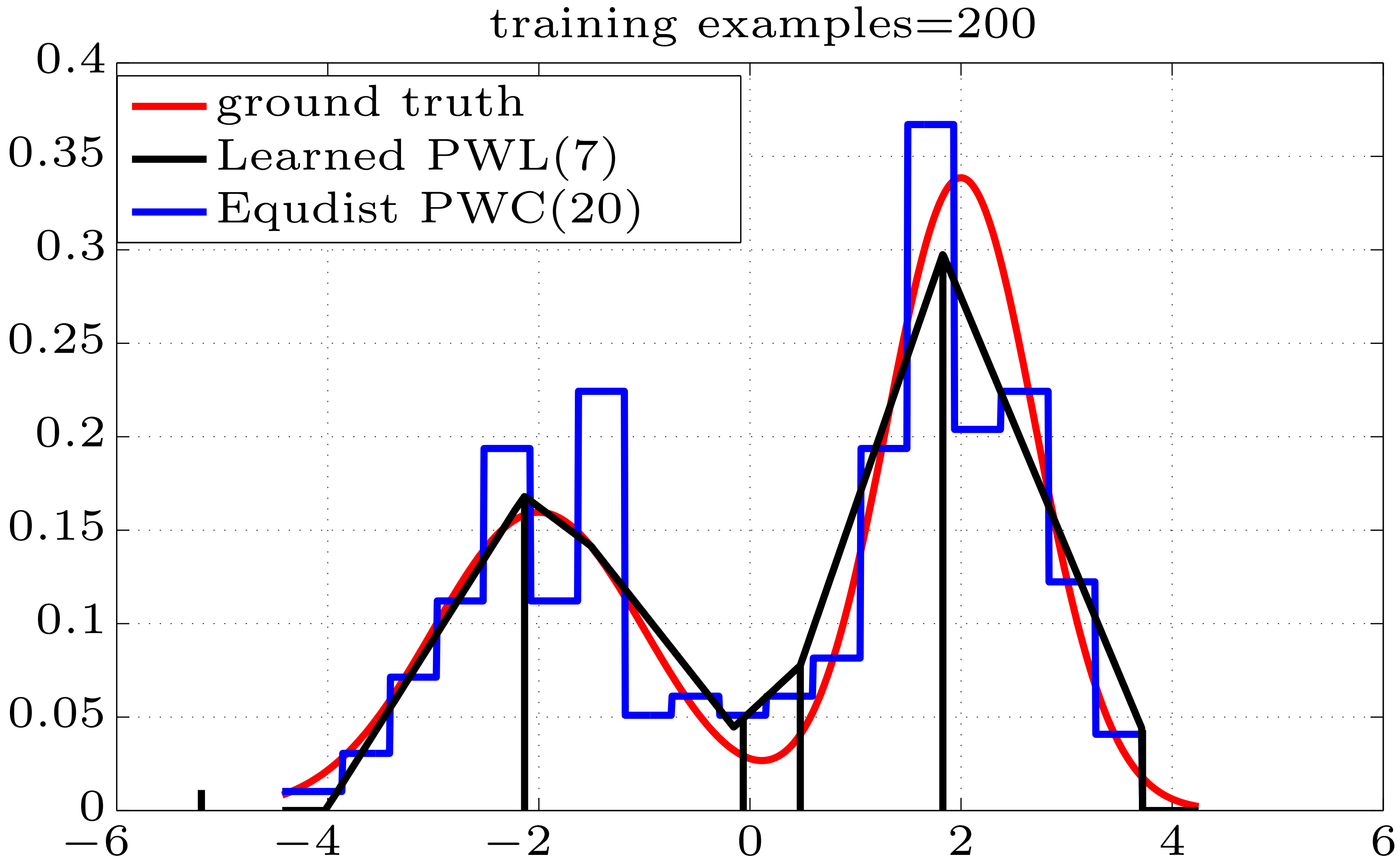
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



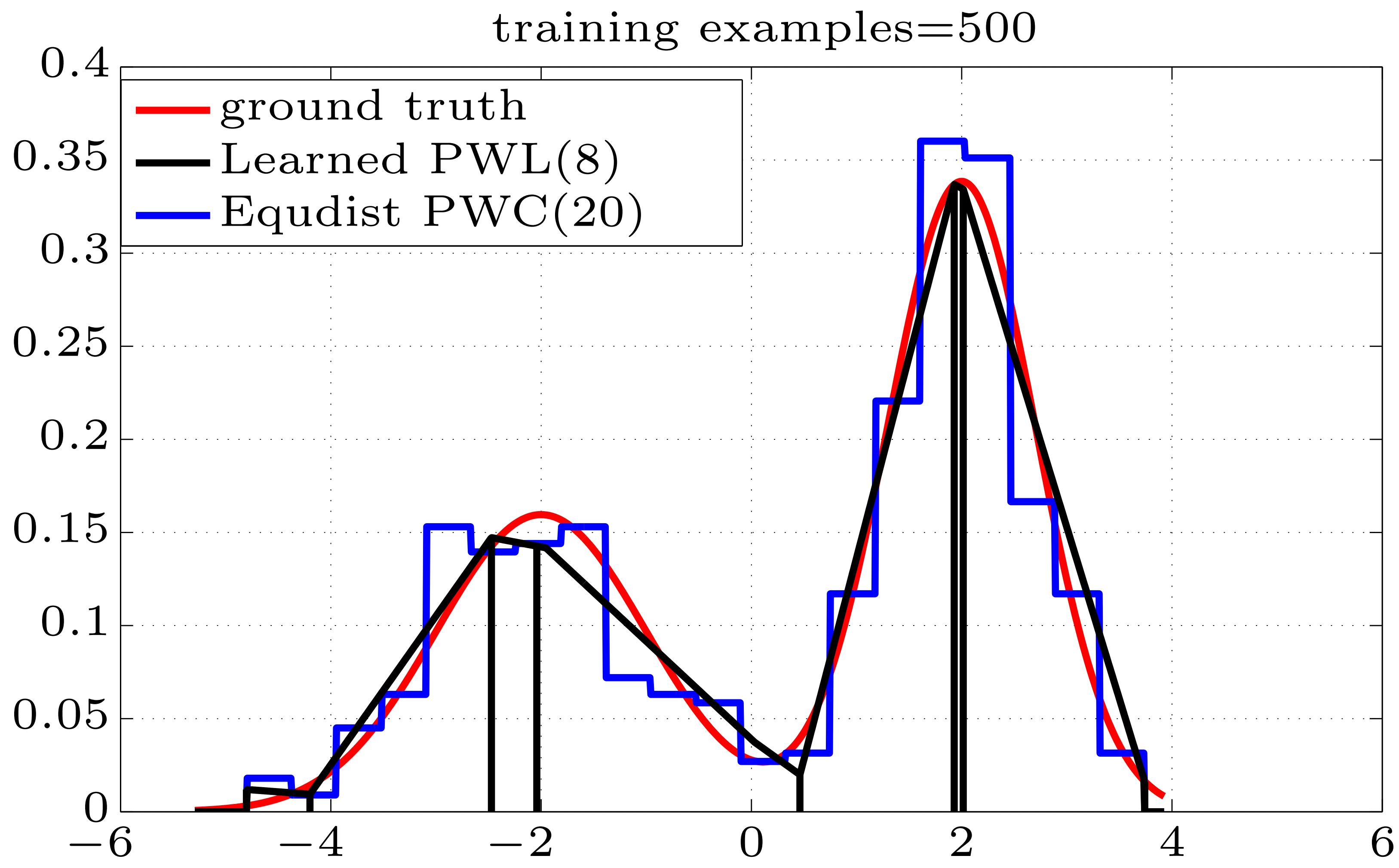
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



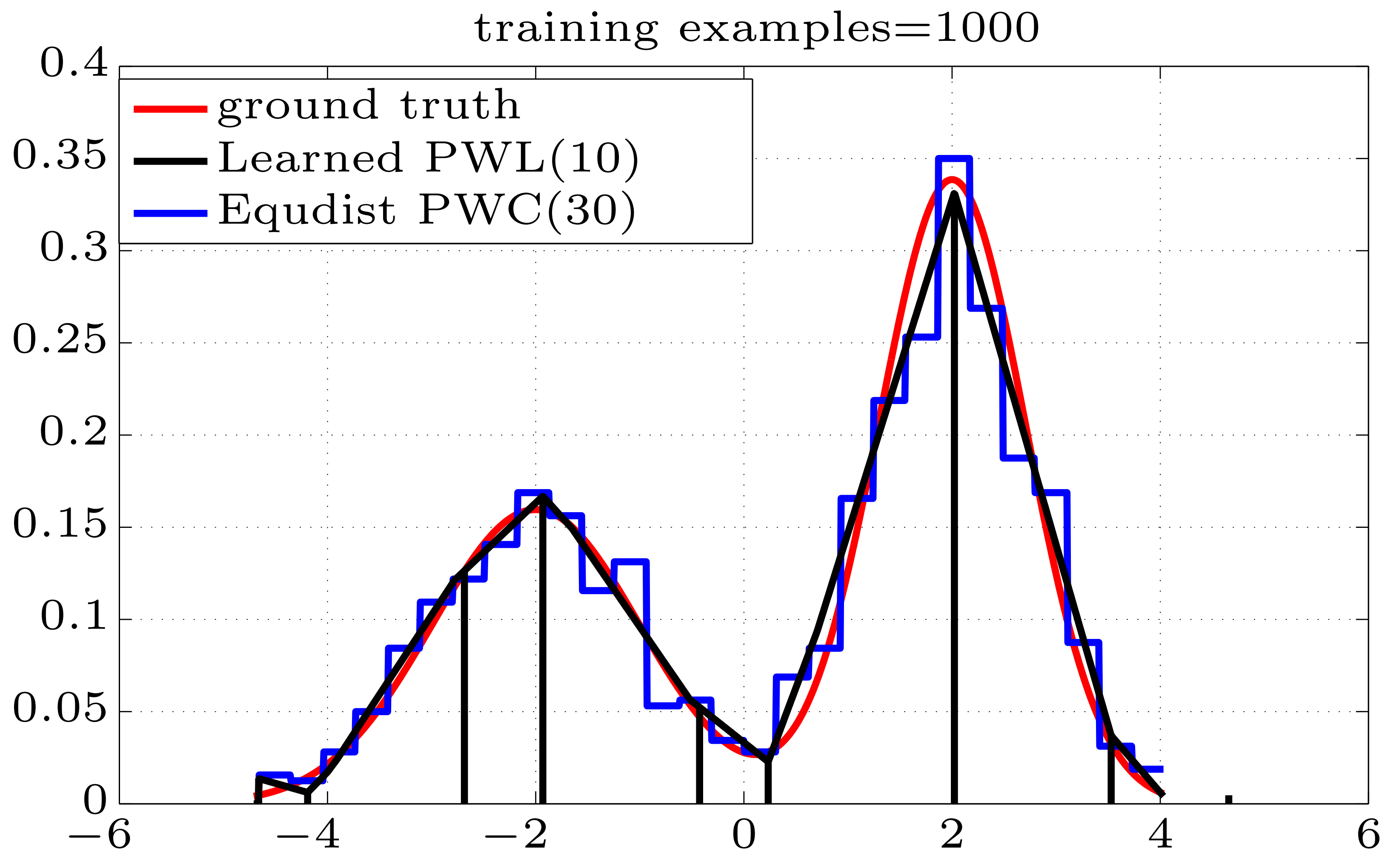
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



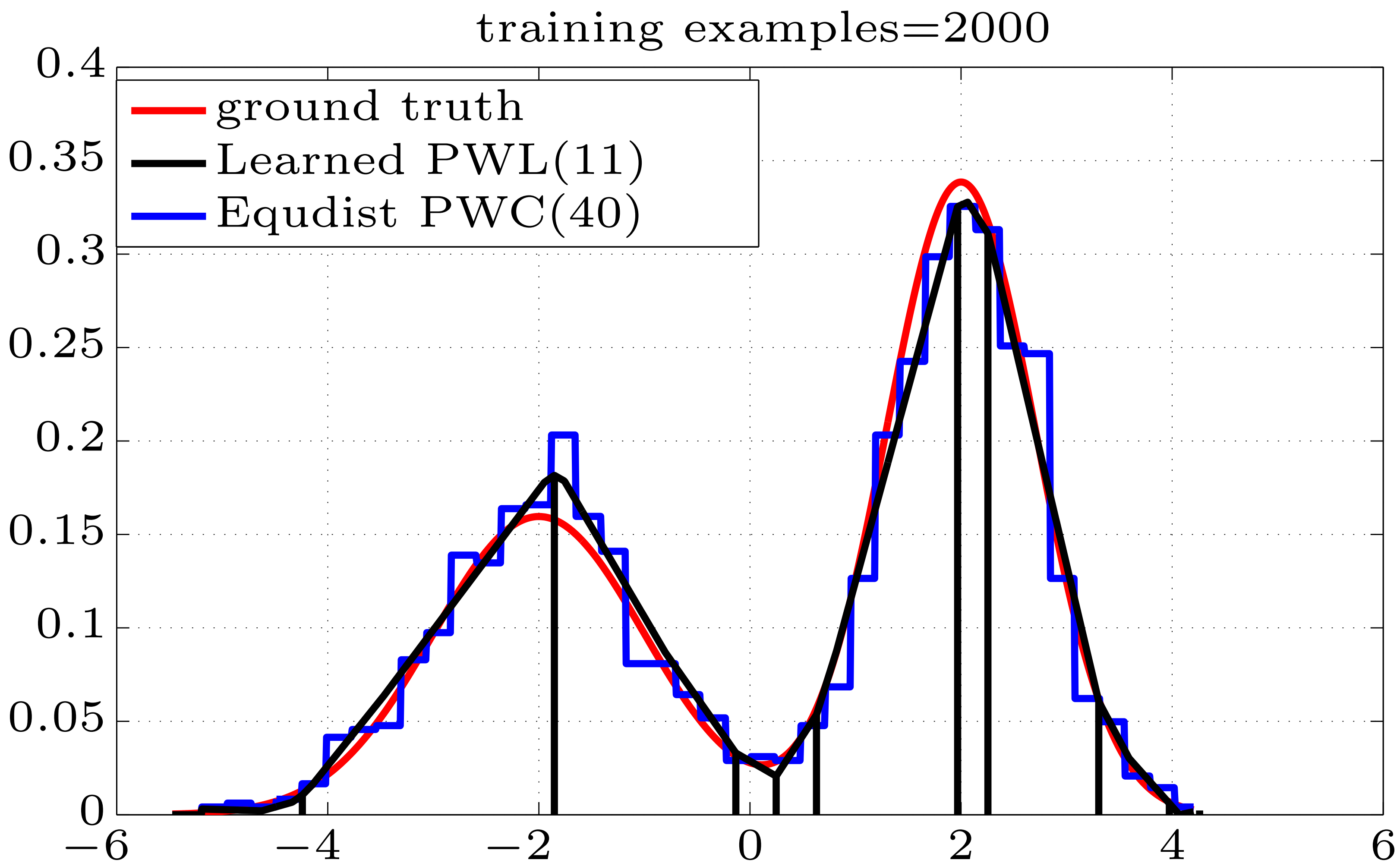
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



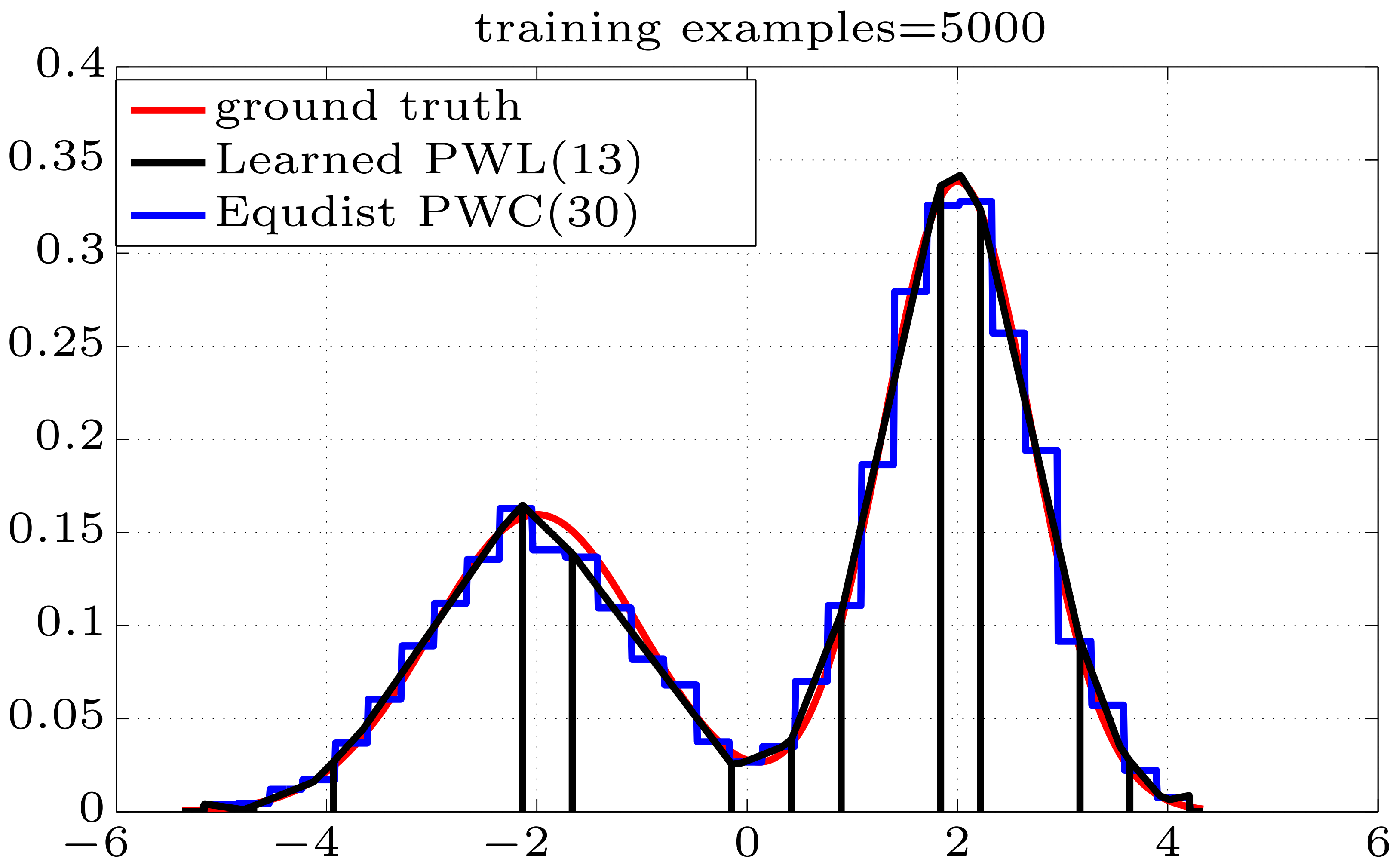
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



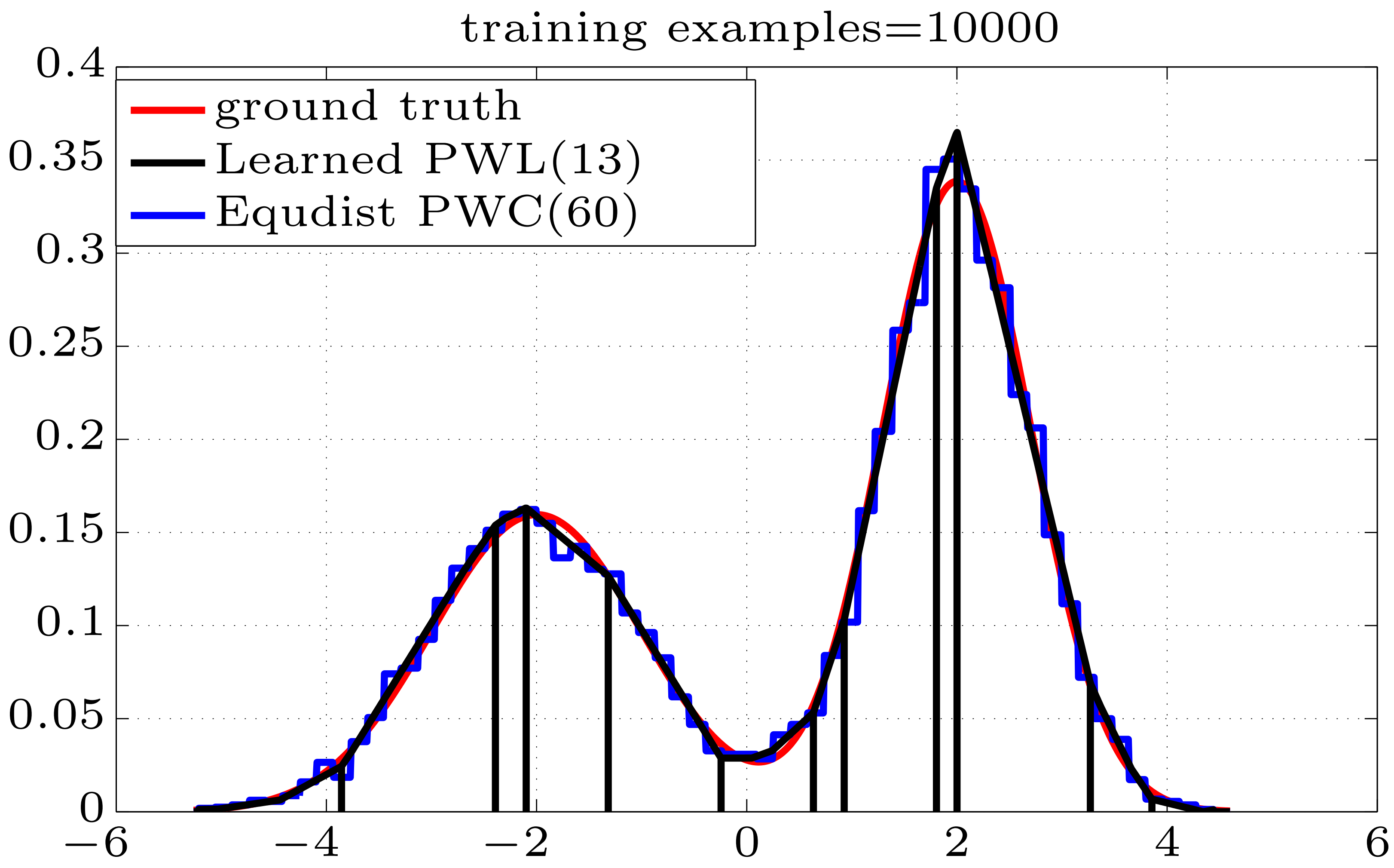
Experiments: learning PWL histograms

- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



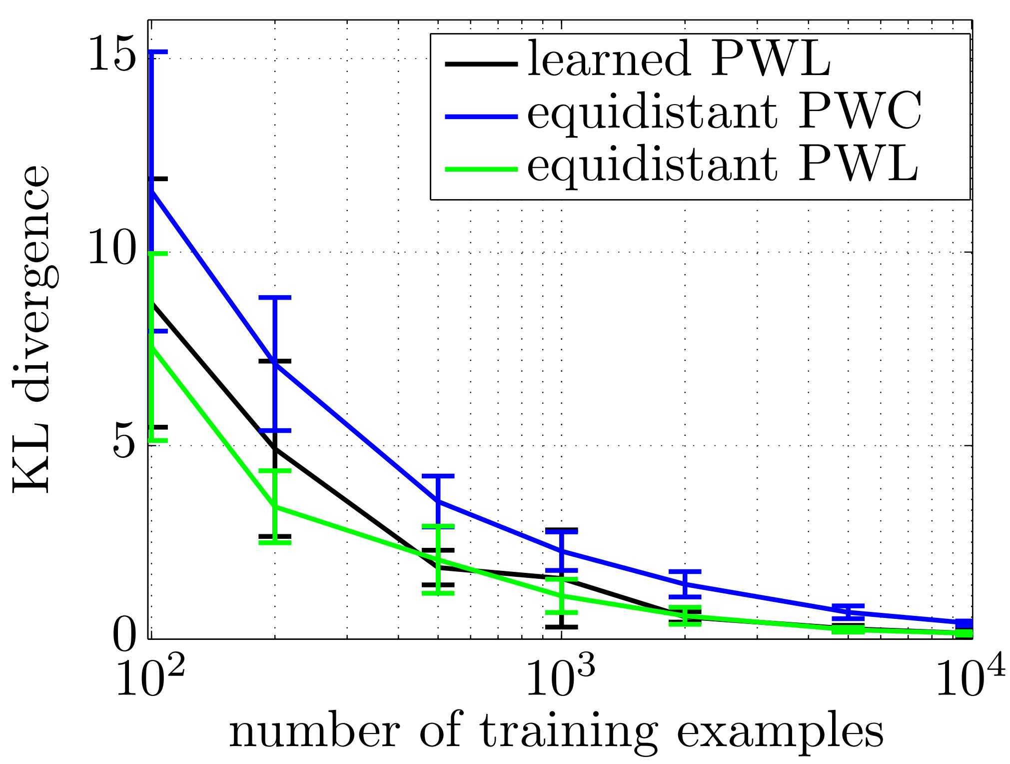
Experiments: learning PWL histograms

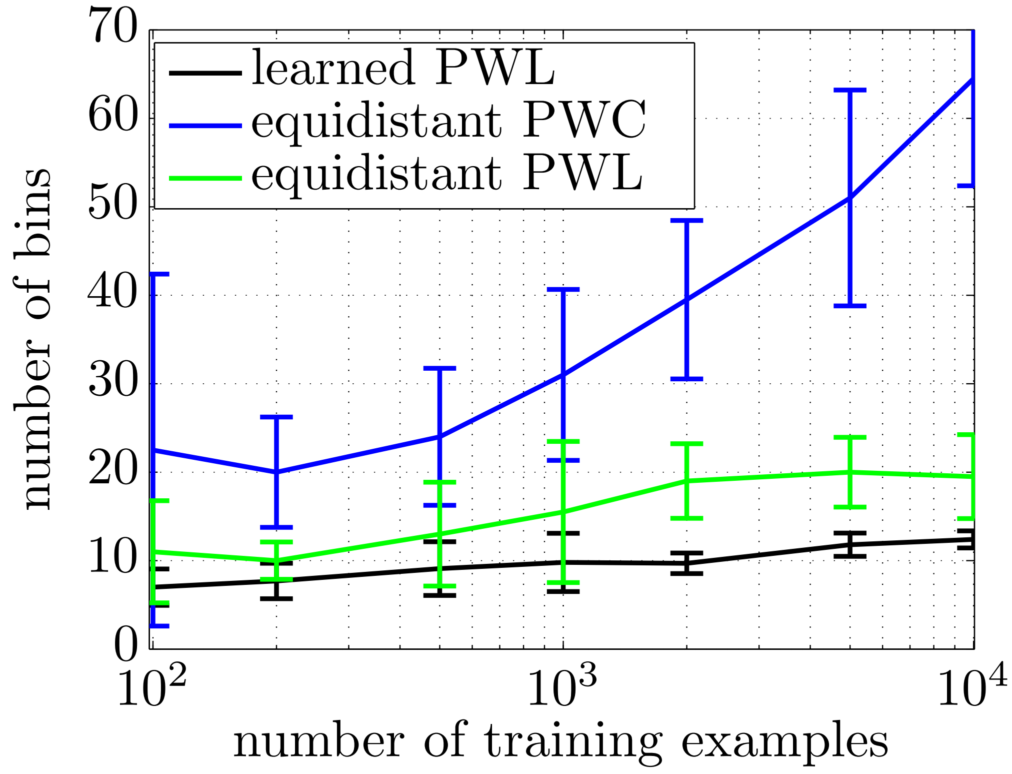
- The initial discretization $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T$ has $D = 100$ equidistant bins.
- The optimal number of bins of PWC histogram selected from $\{5, 10, 20, \ldots, 100\}$.



Experiments: learning PWL histograms

- Comparison of three methods: i) PWL histogram with non-equidistant bins, ii) PWL histogram with equidistant bins and iii) PWC histogram with equidistant bins.
- Methods compared in terms of the KL divergence between the estimated and the true model and the number of bins.
- The optimal number of bins selected based on validation set.





Conclusions

- We propose a generic framework which shows allows to modify a wide class of convex algorithms such that they can learn parameters of PWC and PWL functions.
- The original learning objective is augmented by a convex term which enforces compact bins to emerge from an initial fine discretization.
- In contrast to existing methods we can learn the non-equidistant bins and the weights simultaneously.
- We instantiated the framework for three problems: i) learning PWC histograms for sequence classification, ii) PWL probability density functions and iii) PWL data embedding.
- The empirical evaluation shows that the proposed algorithms yield models with fewer number of parameters and with comparable or better accuracy than the existing ones.

More readings:
`ftp://cmp.felk.cvut.cz/pub/cmp/articles/franc/Franc-TR-2016-01.pdf`