Learning data discretization via convex optimization

Vojtech Franc $\,\cdot\,$ Ondrej Fikar $\,\cdot\,$ Karel Bartos $\,\cdot\,$ Michal Sofka

Received: date / Accepted: date

Abstract Discretization of continuous input functions into piecewise constant or piecewise linear approximations is needed in many mathematical modeling problems. It has been shown that choosing the length of the piecewise segments adaptively based on data samples leads to improved accuracy of the subsequent processing such as classification. Traditional approaches are often tied to a particular classification model which results in local greedy optimization of a criterion function. This paper proposes a technique for learning the discretization parameters along with the parameters of a decision function in a convex optimization of the true objective. The general formulation is applicable to a wide range of learning problems. Empirical evaluation demonstrates that the proposed convex algorithms yield models with fewer number of parameters with comparable or better accuracy than the existing methods.

Keywords piecewise constant embedding \cdot piecewise linear embedding \cdot parameter discretization \cdot convex optimization \cdot classification \cdot histograms

1 Introduction

Many mathematical modeling problems involve discretization of continuous input functions to convert them into their discrete counterparts (García et al., 2013; Liu et al., 2002). The goal of the discretization is to reduce the number of values a continuous attribute assumes by grouping them into a number of predefined bins (Cios et al., 2007, Chapter 8). The discretization is useful for

V. Franc, O. Fikar, K. Bartos, and M. Sofka Czech Technical University in Prague, Faculty of Electrical Engineering Technicka 2, 166 27, Prague 6, Czech Republic E-mail: {xfrancv,fikarond}@cmp.felk.cvut.cz

V. Franc, K. Bartos, and M. Sofka

Cisco systems (Czech Republic), Karlovo namesti 2097/10, 12000 Praha 2, Czech Republic E-mail: {kbartos,msofka}@cisco.com

reducing the data and subsequently generated model complexity and it is a necessary preprocessing step for data mining and machine learning algorithms that operate on discrete attribute spaces (Cios et al., 2007, Chapter 8). The discrete functions are then piece-wise constant (0th order) or piece-wise linear (1st order) approximations of the continuous input functions.

The discretization is useful, for example, when estimating probability density functions. The density functions are typically represented as multi-dimensional histograms in the discrete domain (Silverman, 1986, Chapter 2). Although histograms can asymptotically approximate any continuous density, the accuracy of such approximation depends on the histogram bin size (Rao, 2005, Chapter 2). In another example, the domain of input features is discretized in order to apply a linear decision function as in logistic regression or Support Vector Machine classification (Chapelle et al., 1999). The input feature discretization is a simple method to learn non-linear decision functions where algorithms essentially learn linearly parameterized rules. Clearly, the accuracy of the decision functions directly depends on the discretization of the feature values. One common difficulty in the discretization process is the choice of the discretization step which then indicates the size of the piece-wise segments, e.g. histogram bins, or parameters of the feature representation quantization. The parameters of the decision function are typically estimated in a separate subsequent process (Dougherty et al., 1995; Pele et al., 2013).

Most algorithms employ the simplest unsupervised discretization by choosing a fixed number of bins with the same size (equal interval width) or the same number of samples in each bin (equal frequency interval) (Dougherty et al., 1995). The total number of bins is tuned for a specific application balancing two opposite considerations. Wider bins reduce the effects of noise in regions where the number of input samples is low. On the other hand, narrower bins result in more accurate function approximation in regions where there are many samples and thus the effects of the noise are suppressed. Equal frequency intervals have been extended by using Shannon entropy over discretized space to adjust the bin boundaries (Dougherty et al., 1995). Evidently, using varying bin sizes during discretization can be beneficial.

Supervised discretization algorithms use sample labels to improve the binning (Kerber, 1992; Dougherty et al., 1995), often in an optimization step when learning a classifier (Boullé, 2006; Fayyad and Irani, 1992; Friedman and Goldszmidt, 1996). One widely-adopted approach is to initially start with a large number of bins and then merge neighboring bins while optimizing a criterion function (Boullé, 2006). In (Boullé, 2006), the discretization is based on a search algorithm to find a Bayesian optimal interval splits using a prior distribution on a model space. In (Fayyad and Irani, 1992), the recursive splitting is based on an information entropy minimization heuristic. The algorithm is extended with a Minimum Description Length stopping criteria in (Fayyad and Irani, 1993) and embedded into a dynamic programming algorithm in (Elomaa and Rousu, 1999). These techniques introduce supervision for finding the optimal discretization but are tied to a particular classification model (Naïve Bayes, decision trees, or Bayesian Networks (Friedman and Goldszmidt, 1996; Yang and Webb, 2008)). As a result, they rely on local greedy optimization of the criterion function (Hue and Boullé, 2007).

This paper proposes an algorithm for learning piece-wise constant or piecewise linear embeddings from data samples along with the parameters of a decision function. Similarly to several previous techniques, the initial finegrained discretization with many histogram bins is adjusted by optimizing a criterion function. Our algorithm proposes several important contributions. First, when training a decision function, the algorithm optimizes the true objective (or its surrogate) which includes the discretization parameters and the parameters of the decision function. This is in contrast to previous methods that rely on two separate steps, discretization and classifier learning, which can deviate from the true objective (Pele et al., 2013). Second, the parameter learning is transformed into a convex optimization problem that can be solved effectively. Other techniques do not provide a global solution and resort to a greedy strategy, where the features are processed sequentially (Hue and Boullé, 2007). Third, our formulation is general with piece-wise embeddings being used when training a linear decision function which makes it applicable to a wide range of linear models. Other methods are specific to a particular classification model (Boullé, 2006; Fayyad and Irani, 1992; Friedman and Goldszmidt, 1996; Yang and Webb, 2008).

Our experiments demonstrate that the learned discretization is effective when applied to various data representation problems. The first set of results combines the estimation of the malware traffic representation parameters with learning a linear decision function in a joint convex optimization problem. The learned linear embedding needs much lower dimensionality than equidistant discretization (by a factor of three on average) to achieve the same or better malware detection accuracy level. The second set of results shows a piece-wise linear approximation of a probability density function using nonequidistant bins when estimating the density from data samples. The proposed algorithm achieves lower KL-divergence between the estimate and the ground truth than histograms with equidistant bins. Comparison to the previously published piece-wise linear embedding for non-linear classification (Pele et al., 2013) shows higher accuracy of the proposed technique on a number of datasets from the UCI repository. These encouraging results show the promise of the technique which could be extended to other linear embedding and classification problems.

2 Learning piece-wise constant functions

We consider learning a univariate piece-wise constant (PWC) function

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

where $x \in \mathbb{R}$ is the input variable, *B* is the number of bins, $\boldsymbol{\theta} = (\theta_0, \dots, \theta_B)^T \in \mathbb{R}^{B+1}$ is a vector defining the bin edges, $\boldsymbol{w} = (w_1, \dots, w_B)^T \in \mathbb{R}^B$ is a vector

of weights each of them associated to a single bin and the function

$$k(x, \theta) = 1 + \min\{i \in \{0, \dots, B-1\} \mid x \ge \theta_i\}$$

returns the bin number to which the variable x falls to. For notational convenience, we omit the additive scalar bias w_0 in the definition (1) which does not affect the discussion that follows. We denote set inclusive bracket as '[' and exclusive bracket as ')'. The operator $[\![A]\!]$ evaluates to 1 if the logical statement A is true and it is zero otherwise.

Let $g: \mathbb{R}^m \to \mathbb{R}$ be a convex function whose value $g(f(x^1), \ldots, f(x^m))$ measures how well the function $f: \mathbb{R} \to \mathbb{R}$ evaluated on m training inputs $\mathcal{T} = \{x_i \in \mathbb{R} \mid i = 1, \ldots, m\}$ explains the data. For example, g can be the empirical risk or the likelihood of f evaluated on \mathcal{T} . Assuming the bin edges $\boldsymbol{\theta}$ are fixed, the weights \boldsymbol{w} of the PWC function (1) can be learned by solving the minimization problem

$$\boldsymbol{w}^* \in \underset{\boldsymbol{w} \in \mathbb{R}^B}{\operatorname{argmin}} F_{\operatorname{pwc}}(\boldsymbol{w}, \boldsymbol{\theta})$$
 (2a)

where

$$F_{\text{pwc}}(\boldsymbol{w},\boldsymbol{\theta}) = g(f_{\text{pwc}}(x^1,\boldsymbol{w},\boldsymbol{\theta}),\dots,f_{\text{pwc}}(x^m,\boldsymbol{w},\boldsymbol{\theta}))$$
(2b)

is convex in \boldsymbol{w} since it is a composition of a convex function g and f_{pwc} which is linear in \boldsymbol{w} (Boyd and Vandenberghe, 2004).

In practice the bin edges θ are often selected before learning w. The simplest way is to use equidistantly spaced bins,

$$\theta_i = i(\operatorname{Max} - \operatorname{Min})/B + \operatorname{Min}, \quad \forall i \in \{0, \dots, B\},\$$

where Min and Max are defined such that every admissible input x is inside the interval [Min, Max). The bin edges are constructed for different values of B and the optimal setting is typically tuned on validation examples. This procedure involves minimization of $F_{pwc}(\boldsymbol{w}, \boldsymbol{\theta})$ for all proposal discretizations $\boldsymbol{\theta}$. In principle, one could optimize the width of individual bins as well, however, this would generate exponentially many proposal discretizations making this naive approach intractable.

We propose a method which can simultaneously learn the discretization $\boldsymbol{\theta}$ and the weights \boldsymbol{w} of the PWC function (1). The important feature is that the resulting bins do not have to be equidistant. We only assume that the bin edges are selected from a finite monotonically increasing sequence of real numbers $\nu_0 < \nu_1 < \cdots < \nu_D$, in further text represented by a vector $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_D)^T \in \mathbb{R}^{D+1}$ and denoted as the initial discretization. The set of admissible discretizations $\Theta_B \subset \mathbb{R}^{B+1}$ of the variable $x \in \mathbb{R}$ into B bins contains all vectors $\boldsymbol{\theta}$ satisfying:

$$\theta_i = \nu_{l_i}, i \in \{0, \dots, B\},\tag{3a}$$

$$\begin{cases} l_0 = 0 , \\ l_i < l_{i+1} , i \in \{0, \dots, B-1\} , \\ l_B = D , \end{cases}$$
(3b)

The equation (3a) says that the bin edges $\boldsymbol{\theta} \in \Theta_B$ form a subset of the initial discretization $\boldsymbol{\nu}$. The indices $\{l_0, \ldots, l_B\}$ define which edges are selected from $\boldsymbol{\nu}$. The equations (3b) state that the left most bin edge is ν_0 , the right most edge is ν_D , and the intermediate edges form an increasing sequence.

Given a convex learning algorithm (2), we propose to learn the discretization $\boldsymbol{\theta}^* \in \Theta_B$ simultaneously with the weights $\boldsymbol{w}^* \in \mathbb{R}^B$ by solving

$$(\boldsymbol{w}^*, \boldsymbol{\theta}^*) \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^B, \boldsymbol{\theta} \in \Theta_B} F_{\mathrm{pwc}}(\boldsymbol{w}, \boldsymbol{\theta}).$$
 (4)

Unlike (2) where $\boldsymbol{\theta}$ is fixed, the problem (4) is almost always non-convex and hard to optimize. In the sequel we derive a convex approximation of (4) which can be solved efficiently provided the original problem (2) can be solved efficiently.

Let us define a function $c \colon \mathbb{R}^D \to \{0, \dots, D-1\}$

$$c(v) = \sum_{i=1}^{D-1} [v_i \neq v_{i+1}]$$

returning the total number of different neighboring elements of the vector $\boldsymbol{v} \in \mathbb{R}^{D}$. Let Θ_{B} be induced by the initial discretization $\boldsymbol{\nu}$ as defined by (3). It is easy to see that for any $(\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^{B} \times \Theta_{B})$ there exists a unique $\boldsymbol{v} \in V_{B} = \{\boldsymbol{v} \in \mathbb{R}^{D} | c(\boldsymbol{v}) \leq B-1\}$ such that

$$f_{\text{pwc}}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}) = f_{\text{pwc}}(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\nu}), \quad \forall \boldsymbol{x} \in \mathbb{R}^n,$$
(5)

holds. In particular, \boldsymbol{v} is constructed from $(\boldsymbol{w}, \boldsymbol{\theta})$ by

j

$$v_l = f_{\text{pwc}}(\nu_l, \boldsymbol{w}, \boldsymbol{\theta}), \qquad l \in \{1, \dots, D\}.$$
 (6)

Computing $(\boldsymbol{w}, \boldsymbol{\theta})$ from $(\boldsymbol{v}, \boldsymbol{\nu})$ is also straightforward but not unique in general. It can be seen that for any $\boldsymbol{v} \in V_B$ it is possible to construct a finite number of pairs $\mathcal{W}(\boldsymbol{v}) = \{(\boldsymbol{w}_1, \boldsymbol{\theta}_1), \dots, (\boldsymbol{w}_L, \boldsymbol{\theta}_L)\} \in (\mathbb{R}^B \times \Theta_B)^L$, such that the equality (5) holds for any $(\boldsymbol{w}, \boldsymbol{\theta}) \in \mathcal{W}(\boldsymbol{v})$. Provided $c(\boldsymbol{v}) = B - n$ the conversion from \boldsymbol{v} and $(\boldsymbol{w}, \boldsymbol{\theta})$ is unique, i.e. $|\mathcal{W}(\boldsymbol{v})| = 1$. The parametrization $(\boldsymbol{w}, \boldsymbol{\theta})$ is a shortened representation of $(\boldsymbol{u}, \boldsymbol{\nu})$ which is composed of sub-sequences of the equal components. Therefore we will denote $(\boldsymbol{v}, \boldsymbol{\nu})$ as the uncompressed parametrization and $(\boldsymbol{w}, \boldsymbol{\theta})$ as the compressed parametrization. The equivalence between the compressed parametrization and the uncompressed one is illustrated in Figure 1. The equivalence implies that

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

and therefore the problem (4) can be solved by finding

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v} \in \mathbb{R}^D} F_{\mathrm{pwc}}(\boldsymbol{v}, \boldsymbol{\nu}) \quad \text{s.t.} \quad c(\boldsymbol{v}) \le B - 1,$$
 (7)

and solving (6) for $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ while \boldsymbol{v} is set to \boldsymbol{v}^* . Methods for an efficient computation of the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ from the uncompressed one $(\boldsymbol{v}^*, \boldsymbol{\nu})$ are subject of Section 4.



Fig. 1 The figure shows an example of the PWC function and its two equivalent parametrizations which guarantee that $f_{pwc}(x, \boldsymbol{w}, \boldsymbol{\theta}) = f_{pwc}(x, \boldsymbol{v}, \boldsymbol{\nu}), \forall x \in \mathbb{R}$. The function $f_{pwc}(x, \boldsymbol{w}, \boldsymbol{\theta})$ has 3 bins of heights $\boldsymbol{w} = (w_1, w_2, w_3)^T$ and 4 edges $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_3)^T = (\nu_0, \nu_4, \nu_7, \nu_{12})^T$ selected out of the sequence $\boldsymbol{\nu} = (\nu_0, \dots, \nu_{12})^T$. The function $f_{pwc}(x, \boldsymbol{v}, \boldsymbol{\nu})$ has 11 bins with 12 edges $\boldsymbol{\nu}$ and heights $\boldsymbol{v} = (w_1, w_1, w_1, w_2, w_2, w_2, w_3, w_3, w_3, w_3, w_3)^T$. Though \boldsymbol{v} has 11 components there are only two places where they differ, hence $\boldsymbol{c}(\boldsymbol{v}) = 2$.

The problem (7) has a convex objective but its single constraint remains non-convex. By introducing a vector $\boldsymbol{d} = (v_1 - v_2, \dots, v_{D-1} - v_D)^T$ we see that the function $c(\boldsymbol{v})$ can be written as the L_0 -norm of the vector $\boldsymbol{d} = (v_1 - v_2, \dots, v_{D-1} - v_D)^T$, i.e. $c(\boldsymbol{v}) = \|\boldsymbol{d}\|_0$. It has been demonstrated that the L_1 -norm is often a good convex proxy of the L_0 -norm, see e.g. (Candes and Tao, 2005; Candes et al., 2006; Donoho, 2006). Therefore, we propose to approximate the non-convex function $c(\boldsymbol{v}) = \|\boldsymbol{v}\|_0$ by the convex function $\tilde{c}(\boldsymbol{v}) = \|\boldsymbol{d}\|_1$. A convex relaxation of the problem (7) then reads

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

or, equivalently, we can solve an unconstrained problem

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v} \in \mathbb{R}^D} \left[F_{\mathrm{pwc}}(\boldsymbol{v}, \boldsymbol{\nu}) + \gamma \sum_{i=1}^{D-1} |v_i - v_{i+1}| \right],$$
(8)

where the constant $\gamma \in \mathbb{R}_+$ is monotonically dependent on B-1.

3 Learning piece-wise linear functions

The previous section shows how to transform learning of PWC functions to a convex optimization problem. A similar idea can be applied to learning piecewise linear (PWL) functions which provide more accurate approximation of the input data. In this case, we want to learn a function

$$f_{\text{pwl}}(x, \boldsymbol{w}, \boldsymbol{\theta}) = w_{k(x, \boldsymbol{\theta}) - 1} \cdot (1 - \alpha(x, \boldsymbol{\theta})) + w_{k(x, \boldsymbol{\theta})} \cdot \alpha(x, \boldsymbol{\theta})$$
(9)



Fig. 2 The figure shows an example of the PWL function and its two equivalent parametrizations which guarantee that $f_{pwl}(x, \boldsymbol{w}, \boldsymbol{\theta}) = f_{pwl}(x, \boldsymbol{u}, \boldsymbol{\nu}), \forall x \in \mathbb{R}$. The function $f_{pwl}(x, \boldsymbol{w}, \boldsymbol{\theta})$ is described by 3 line segments connecting 4 points $\{(\theta_0, w_0), \ldots, (\theta_3, w_3)\}$ whose coordinates are stored in $\boldsymbol{w} = (w_0, \ldots, w_3)^T$ a $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_3)^T$. The bin edges $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \theta_3)^T = (\nu_0, \nu_4, \nu_7, \nu_{12})^T$ form a subset of $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_{12})^T$. The function $f_{pwl}(x, \boldsymbol{u}, \boldsymbol{\nu})$ is described by 11 line segments connecting points $\{(\nu_0, u_0), \ldots, (\nu_{12}, u_{12})\}$ whose coordinates are stored in $\boldsymbol{u} = (u_0, \ldots, u_{12})^T$ and $\boldsymbol{\nu}$. Note, however, that only 4 components of \boldsymbol{u} cannot be expressed as an average of its neighbors, hence $e(\boldsymbol{u}) = 4$.

where $x \in \mathbb{R}$ is the input variable as for before, $\boldsymbol{\theta} \in \mathbb{R}^{B+1}$ is a vector defining the bin edges, B is the number of bins and $\boldsymbol{w} \in \mathbb{R}^{B+1}$ is the vector of weights ¹. The function $\alpha \colon \mathbb{R} \times \mathbb{R}^{B+1} \to [0, 1]$, defined as

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

is a normalized distance between x and the right edge of the $k(x, \theta)$ -th bin.

Analogously to the previous section, we want to learn simultaneously the discretization $\theta^* \in \Theta_B$ and the weights $w^* \in \mathbb{R}^{B+1}$ by solving

$$(\boldsymbol{w}^*, \boldsymbol{\theta}^*) \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^{B+1}, \boldsymbol{\theta} \in \Theta_B} F_{\mathrm{pwl}}(\boldsymbol{w}, \boldsymbol{\theta})$$
 (10)

where

$$F_{\text{pwl}}(\boldsymbol{w}, \boldsymbol{\theta}) = g(f_{\text{pwl}}(x^1, \boldsymbol{w}, \boldsymbol{\theta}), \dots, f_{\text{pwl}}(x^m, \boldsymbol{w}, \boldsymbol{\theta}))$$

is the learning objective depending on the responses of the PWL function (9) and evaluated on the training inputs $\{x^1, \ldots, x^m\} \in \mathbb{R}^m$. In the sequel, we derive a convex relaxation of (10).

Let us define a function $e: \mathbb{R}^{D+1} \to \{0, \dots, D+1\},\$

$$e(\boldsymbol{u}) = \sum_{i=1}^{D-1} \llbracket u_i \neq \frac{1}{2} (u_{i-1} + u_{i+1}) \rrbracket$$

which returns the number of weights that cannot be expressed as an average of the neighboring weights. Let Θ_B be induced by the initial discretization ν

 $^{^1\,}$ Note that the PWC function has B weights associated with the bins while the PWL function has B+1 weights associated with the bin edges.

as defined by (3). It can be seen that for any $(\boldsymbol{w}, \boldsymbol{\theta}) \in (\mathbb{R}^{B+1} \times \Theta_B)$ we can construct a unique $\boldsymbol{u} \in \mathcal{U}_B = \{\boldsymbol{u} \in \mathbb{R}^{D+1} \mid e(\boldsymbol{u}) \leq B-1\}$ such that

$$f_{\text{pwl}}(x, \boldsymbol{w}, \boldsymbol{\theta}) = f_{\text{pwl}}(x, \boldsymbol{u}, \boldsymbol{\nu}), \quad \forall x \in \mathbb{R}^n$$
 (11)

holds, in particular, \boldsymbol{u} is constructed from $(\boldsymbol{w}, \boldsymbol{\theta})$ by

$$u_l = f_{pwl}(\nu_l, \boldsymbol{w}, \boldsymbol{\theta}), \quad l \in \{0, \dots, D\}.$$
(12)

In addition, for any $\boldsymbol{u} \in \mathcal{U}_B$ it is possible to construct a finite number of pairs $\mathcal{W}'(\boldsymbol{u}) = \{(\boldsymbol{w}_1, \boldsymbol{\theta}_1), \cdots, (\boldsymbol{w}_L, \boldsymbol{\theta}_L)\} \in (\mathbb{R}^{B+1} \times \Theta_B)^L$ such that the equality (11) holds for any $(\boldsymbol{w}, \boldsymbol{\theta}) \in \mathcal{W}'(\boldsymbol{u})$. For $e(\boldsymbol{u}) = B - 1$ the conversion from \boldsymbol{u} to $(\boldsymbol{w}, \boldsymbol{\theta})$ is unique, i.e. $|\mathcal{W}'(\boldsymbol{u})| = 1$. The equivalence between the compressed parametrization $(\boldsymbol{w}, \boldsymbol{\theta})$ and the uncompressed parametrization $(\boldsymbol{u}, \boldsymbol{\nu})$ is illustrated in Figure 2. The equivalence implies that

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

Consequently, the problem (10) can be solved by choosing

$$\boldsymbol{u}^* \in \operatorname*{argmin}_{\boldsymbol{u} \in \mathbb{R}^{D+1}} F_{\mathrm{pwl}}(\boldsymbol{u}, \boldsymbol{\nu}) \quad \mathrm{s.t.} \quad e(\boldsymbol{u}) \le B - 1 ,$$
 (13)

and computing $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ from \boldsymbol{u}^* by solving the equation (12) for $(\boldsymbol{w}, \boldsymbol{\theta})$ with $\boldsymbol{u} = \boldsymbol{u}^*$. Efficient methods computing the compressed parametrization $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ from the uncompressed one $(\boldsymbol{u}^*, \boldsymbol{\nu})$ are discussed in Section 4. As before, replacing the L_0 -norm in the non-convex constraint $e(\boldsymbol{u}) = B-1$ by the L_1 -norm, we obtain a convex relaxation of (10) which reads

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

or, equivalently, we can solve an unconstrained problem

$$\boldsymbol{u}^* \in \operatorname*{argmin}_{\boldsymbol{u} \in \mathbb{R}^{D+1}} \left[F_{\text{pwl}}(\boldsymbol{u}, \boldsymbol{\nu}) + \gamma \sum_{i=1}^{D-1} \left| u_i - \frac{1}{2} u_{i-1} - \frac{1}{2} u_{i+1} \right| \right].$$
(14)

where the constant $\gamma \in \mathbb{R}_+$ is monotonically dependent on B-1.

4 Rounding of piece-wise functions

The uncompressed parametrization $(\boldsymbol{v}, \boldsymbol{\nu})$ of the PWC function (e.g. found by the proposed algorithm (8)) can be converted to the compressed one $(\boldsymbol{w}, \boldsymbol{\theta})$ by splitting \boldsymbol{v} into sub-vectors of equal weights, i.e. $\boldsymbol{v} = (\boldsymbol{v}_1^T, \boldsymbol{v}_2^T, \dots, \boldsymbol{v}_B^T)$ where each \boldsymbol{v}_i can be written as $\boldsymbol{v}_i = w_i[1, \dots, 1]^T$. Obtaining the compressed parametrization $(\boldsymbol{w}, \boldsymbol{\theta})$ is then straightforward (see Figure 1). An analogous procedure can be applied for the PWL parametrizations in which case we are searching for sub-vectors whose intermediate components can be expressed as an average of its neighbors. In practice, however, the components of the uncompressed solution can be noisy thanks to the used convex relaxation and usage of approximate solvers to find the uncompressed parameters. For this reason, it is useful to round the uncompressed solution before its conversion to the compressed one. The rounding procedures for the PWC and the PWL parametrization are described in the next sections.

4.1 Rounding of PWC parametrization

A very crude rounding is obtained by splitting \boldsymbol{v} into the sub-vectors $\boldsymbol{v} = (\boldsymbol{v}_1^T, \boldsymbol{v}_2^T, \dots, \boldsymbol{v}_B^T)$ such that all components of each \boldsymbol{v}_i have the same sign. The rounded uncompressed solution is then

$$\bar{\boldsymbol{v}} = (\operatorname{mean}(\boldsymbol{v}_1)[1,\ldots,1]^T,\ldots,\operatorname{mean}(\boldsymbol{v}_B)[1,\ldots,1]^T)^T$$
(15)

where the operator mean(v') returns the mean value of the components of the vector v'. Note that this procedure does not require any parameter.

Another method is to find the parameters \bar{v} of the PWC function with B bins which have the shortest Euclidean distance to the given v by solving

$$\bar{\boldsymbol{v}} \in \underset{\boldsymbol{v}' \in \mathbb{R}^D}{\operatorname{argmin}} \|\boldsymbol{v} - \boldsymbol{v}'\|^2 \quad \text{s.t.} \quad c(\boldsymbol{v}') = B - 1.$$
 (16)

The problem (16) is the Euclidean projection of \boldsymbol{v} onto the set $V_B = \{\boldsymbol{v} \in \mathbb{R}^D | c(\boldsymbol{v}) \leq B-1\}$. The optimal solution of (16) can be found by dynamic programming as follows. Provided that $c(\boldsymbol{v}') = B-1$ holds, the vector \boldsymbol{v}' can be parametrized by a shorter vector $\boldsymbol{w} = (w_1, \ldots, w_B) \in \mathbb{R}^b$ and indices (l_0, \ldots, l_B) satisfying (3b) which define bins and hence also sub-sequences within \boldsymbol{v}' with the same value, i.e. $v'_{l_i+1} = \cdots = v'_{l_i} = w_i$ holds for all $i \in \{1, \ldots, B\}$. With this notation, we can rewrite the distance $\|\boldsymbol{v} - \boldsymbol{v}'\|^2$ as

$$\|\boldsymbol{v} - \boldsymbol{v}'\|^2 = \sum_{i=1}^{D} (v_i - v_i')^2 = \sum_{j=1}^{B} \sum_{i=l_{j-1}+1}^{l_j} (v_i - w_j)^2 = \sum_{j=1}^{B} R(l_{j-1}, l_j, w_j) .$$

It is clear that weight \bar{w}_j solving the sub-problem $\min_{w_j} R(l_{j-1}, l_j, w_j)$ equals the arithmetic mean of the corresponding weights, $\bar{w}_j = \sum_{i=l_{j-1}+1}^{l_j} v_i/(l_j - l_{j-1})$. Let us define $\hat{R}(l_{j-1}, l_j) = R(l_{j-1}, l_j, \hat{w}_j)$. Instead of solving (16) we can now search for the optimal indices:

$$(\bar{l}_0, \dots, \bar{l}_B) \in \underset{(l_0, \dots, l_B)}{\operatorname{argmin}} \sum_{j=1}^B \hat{R}(l_{j-1}, l_j) \quad \text{s.t} \quad \begin{array}{l} l_0 = 0, \\ l_i < l_{i+1}, i \in \{0, \dots, B-1\} \\ l_B = D. \end{array}$$
(17)

The problem (17) can be solved by dynamic programming since its objective decomposes into a sum of functions each of which sharing at most two variables with other functions. The overall computational time of the DP procedure is $\mathcal{O}(B \cdot D^2)$.

Because the number of bins is often unknown a priori, we can search for the minimal number of bins which explain the given solution v with a prescribed precision ε , i.e. we can solve

$$\bar{\boldsymbol{v}} = \operatorname*{argmin}_{B \in \mathcal{N}, \boldsymbol{v}' \in \mathbb{R}^D} B \quad \text{s.t.} \quad \|\boldsymbol{v} - \boldsymbol{v}'\|^2 \le \varepsilon \quad \text{and} \quad c(\boldsymbol{v}') = B - 1.$$
(18)

The solution of (18) can be converted to the solution of (16) by increasing B until the constraint $\|\boldsymbol{v} - \boldsymbol{v}'\|^2 \leq \varepsilon$ is satisfied.

4.2 Rounding PWL function

The parameter vector $\bar{\boldsymbol{u}}$ describing the PWL function with *B* bins which has the shortest Euclidean distance to given \boldsymbol{u} can be found by solving

$$\bar{\boldsymbol{u}} \in \underset{\boldsymbol{u}' \in \mathbb{R}^{D+1}}{\operatorname{argmin}} \|\boldsymbol{u} - \boldsymbol{u}'\|^2 \quad \text{s.t.} \quad e(\boldsymbol{u}') = B - 1.$$
(19)

If the number of bins is unknown, we can search for the minimal number of bins which explain the given solution u with a prescribe precision ε by solving

$$\bar{\boldsymbol{u}} = \underset{B \in \mathcal{N}, \boldsymbol{u}' \in \mathbb{R}^{D+1}}{\operatorname{argmin}} B \quad \text{s.t.} \quad \|\boldsymbol{u} - \boldsymbol{u}'\|^2 \le \varepsilon \quad \text{and} \quad e(\boldsymbol{v}') = B - 1.$$
(20)

The problems (19) and (20) can be solved exactly by procedures analogous to the ones for the PWC function which were described in previous section and hence omitted for the sake of space.

5 Examples of the proposed framework

The previous sections desribe a generic framework that allows to modify a wide class of convex algorithms so that they can learn the PWC and the PWL functions. In this section, we give three instances of the proposed framework. We also show how the same idea can be applied to learning multi-variate PWC and PWL functions. In particular, we consider learning of the linear classifiers using sequential data represented by PWC histograms (Section 5.1), estimation of the PWL probability density models (Section 5.2) and learning of the non-linear classifiers via the PWL data embedding (Section 5.3). It will be shown that learning leads to an instance of convex optimization problem in all cases. Moreover, the optimization can be reformulated as a well-understood Quadratic Programming task.

5.1 Classification of histograms

In many applications, the object to be classified is described by a set of sequences sampled from some unknown distributions. A simple yet effective representation of the sequential data is the normalized PWC histogram which is then used as an input to a linear classifier. This classification model has been successfully used e.g. in computer vision (Dalal and Triggs, 2005) or in the computer security (Bartos and Sofka, 2015) as will be described in Section 7.1.

Assume we have a training set $\mathcal{T} = \{(\mathbf{X}^1, y^1), \dots, (\mathbf{X}^m, y^m)\} \in (\mathbb{R}^{n \times d} \times \{-1, +1\})^m$ where the input matrix \mathbf{X}^i describes *n* sequences each having *d* elements. The linear classifier $h(\mathbf{X}, \mathbf{w}, \boldsymbol{\theta}) = \text{sign}(f_{\text{pwc}}(\mathbf{X}, \mathbf{w}, \boldsymbol{\theta}))$ assigns \mathbf{X} into a class based on the sign of the discriminant function

$$f_{\text{pwc}}(\boldsymbol{X}, \boldsymbol{w}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{j=1}^{b_i} \frac{1}{d} \sum_{k=1}^{d} \llbracket X_{i,k} \in [\theta_{i,j-1}, \theta_{i,j}) \rrbracket w_{i,j}$$
(21)

where $\boldsymbol{\theta}_i = (\theta_{i,0}, \dots, \theta_{i,b_i})^T \in \mathbb{R}^{b_i+1}$ is a vector defining bin edges of the *i*-th histogram and $\boldsymbol{\theta} = (\boldsymbol{\theta}_1^T, \dots, \boldsymbol{\theta}_n^T)^T \in \mathbb{R}^{n+B}$ is their concatenation, $B = \sum_{i=1}^n b_i$ denotes the total number of bins, $\boldsymbol{w}_i = (w_{i,1}, \dots, w_{i,b_i})^T \in \mathbb{R}^{b_i}$ are bin heights of the *i*-th histogram and $\boldsymbol{w} = (\boldsymbol{w}_1^T, \dots, \boldsymbol{w}_n^T) \in \mathbb{R}^B$ is their concatenation.

In order to learn the bin edges $\boldsymbol{\theta}$ and the weights \boldsymbol{w} from examples \mathcal{T} simultaneously, we apply the general framework described in Section 2. In particular, we show how to adapt the Support Vector Machine (SVM) algorithm. First, we define the initial discretization $\boldsymbol{\nu}_i = (\nu_{i,0}, \ldots, \nu_{i,D})^T \in \mathbb{R}^{D+1}$ for each of the *n* histograms. For example, we place D + 1 edges equidistantly between the minimal Min_i and the maximal Max_i value that can appear in the *i*-th sequence, so that $\nu_{i,j} = j(\text{Max}_i - \text{Min}_i) + \text{Min}_i$. Second, we combine the SVM objective function

$$F_{\text{pwc}}^{\text{svm}}(\boldsymbol{w}, \boldsymbol{\theta}, \lambda) = \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \max\left\{0, 1 - y_i f_{\text{pwc}}(\boldsymbol{x}^i, \boldsymbol{w}, \boldsymbol{\theta})\right\}$$

with the L_1 -norm approximation of the function

$$c_n(v) = \sum_{i=1}^n \sum_{j=1}^{D-1} [v_{i,j} \neq v_{i,j+1}]$$

The function $c_n(\boldsymbol{v})$ indicates the total number of different neighboring components of \boldsymbol{v} . Analogously to the general formulation (8), we obtain the following convex program

$$\boldsymbol{v}^* \in \operatorname*{argmin}_{\boldsymbol{v} \in \mathbb{R}^{nD}} \left[F_{\mathrm{pwc}}^{\mathrm{svm}}(\boldsymbol{v}, \boldsymbol{\nu}, \lambda) + \gamma \sum_{i=1}^n \sum_{j=1}^{D-1} |v_{i,j} - v_{i,j+1}| \right].$$
(22)

The constant $\lambda \in \mathbb{R}_+$ controls the empirical error. The constant $\gamma \in \mathbb{R}_+$ influences the number of equal neighboring in v^* and thus the number of

emerging bins. The uncompressed parametrization $(\boldsymbol{v}^*, \boldsymbol{\nu})$ is converted to the compressed one $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ via the rounding methods from Section 4, which are applied to each pair $(\boldsymbol{v}_i^*, \boldsymbol{\nu}_i), i \in \{1, \ldots, n\}$, separately.

5.2 Estimation of PWL histograms

Given a training sample $\mathcal{T} = \{x^i \in \mathbb{R} \mid i = 1, ..., m\}$ drawn from i.i.d. random variables with unknown distribution p(x), the goal is to find $\hat{p}(x)$ approximating p(x) accurately based on the samples \mathcal{T} . Assume we want to model the unknown p.d.f. by the PWL function

$$\hat{p}_{\text{pwl}}(x, \boldsymbol{w}, \boldsymbol{\theta}) = \left(1 - \alpha(x, \boldsymbol{\theta})\right) w_{k(x, \boldsymbol{\theta}) - 1} + \alpha(x, \boldsymbol{\theta}) w_{k(x, \boldsymbol{\theta})}, \tag{23}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{B+1}$ are the bin edges and $\boldsymbol{w} \in \mathbb{R}^{B+1}_+$ is a vector of non-negative weights selected such that $\int \hat{p}_{pwl}(x, \boldsymbol{w}, \boldsymbol{\theta}) dx = 1$. To learn the unknown parameters $(\boldsymbol{w}, \boldsymbol{\theta})$ from the sample \mathcal{T} , we are going to instantiate the framework proposed in Section 3 to the Maximum Likelihood method. Let the initial discretization $\boldsymbol{\nu} \in \mathbb{R}^{D+1}$ be equidistantly spaced between the minimal and the maximal value, i.e. $\nu_j = j(\text{Max} - \text{Min}) + \text{Min}, \forall j \in \{0, \dots, D\}$, where D is set to be sufficiently high. We substitute the negative log-likelihood

$$F_{\text{pwl}}^{\text{nll}}(\boldsymbol{w}, \boldsymbol{\theta}) = -\sum_{i=1}^{m} \log \hat{p}_{\text{pwl}}(x^{i}, \boldsymbol{w}, \boldsymbol{\theta})$$

to the general formulation (14) which yields the following convex problem

$$\boldsymbol{u}^* = \operatorname*{argmin}_{\boldsymbol{u} \in \mathbb{R}^D} \left[F_{\mathrm{pwl}}^{\mathrm{nnl}}(\boldsymbol{u}, \boldsymbol{\nu}) + \gamma \sum_{j=1}^{D-1} \left| u_j - \frac{1}{2} u_{j-1} - \frac{1}{2} u_{j+1} \right| \right]$$
(24a)

subject to

$$u_0 + u_D + 2\sum_{i=1}^{D-1} u_i = \frac{2D}{\text{Max} - \text{Min}}, \qquad u_i \ge 0, \quad i \in \{0, \dots, D\}, \quad (24b)$$

where $\gamma \in \mathbb{R}_+$ is a constant controlling the number of bins. The introduced constraints (24b) ensure that $\int_{\text{Min}}^{\text{Max}} \hat{p}_{\text{pwl}}(x, \boldsymbol{u}, \boldsymbol{\nu}) dx = 1$ holds for any feasible \boldsymbol{u} . The found \boldsymbol{u}^* defines a PWL probability density model $\hat{p}_{\text{pwl}}(x^i, \boldsymbol{u}^*, \boldsymbol{\nu})$. If necessary, the compressed parameters $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$, defining the non-equidistant bins, can be recovered from \boldsymbol{u}^* by the rounding methods described in Section 4.

5.3 PWL embedding for non-linear classification

The PWL embedding (Pele et al., 2013) is a simple yet efficient way to learn highly non-linear decision function by a linear algorithm such as the Support Vector Machines. Let $\mathcal{T} = \{(\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^m, y^m)\} \in (\mathbb{R}^n \times \{-1, +1\})^m$ be a training set of input features and output binary labels. A linear classifier $h(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}) = \operatorname{sign}(f_{\operatorname{pwl}}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}))$ assigns the input $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ into classes based on the sign of the discriminant function

$$f_{\text{pwl}}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \left(w_{i,k(x_i, \boldsymbol{\theta}_i)-1} \cdot \left(1 - \alpha(x_i, \boldsymbol{\theta}_i)\right) + w_{i,k(x_i, \boldsymbol{\theta}_i)} \cdot \alpha(x_i, \boldsymbol{\theta}_i) \right)$$
(25)

which is a sum of *n* PWL functions each defined for a single input feature. The vector $\boldsymbol{\theta}_i \in \mathbb{R}^{b_i+1}$ contains the bin edges of the *i*-th feature and $\boldsymbol{\theta} = (\boldsymbol{\theta}_1^T, \dots, \boldsymbol{\theta}_n^T)^T \in \mathbb{R}^{n+B}$ is their concatenation where $B = \sum_{i=1}^n b_i$ denotes the total number of bins. The vector $\boldsymbol{w}_i \in \mathbb{R}^{b_i+1}$ contains the weights associated to the edges $\boldsymbol{\theta}_i$ and $\boldsymbol{w} = (\boldsymbol{w}_1^T, \dots, \boldsymbol{w}_n^T) \in \mathbb{R}^{B+n}$ is their concatenation.

We can learn the weights \boldsymbol{w} as well as the discretization $\boldsymbol{\theta}$ using the SVM algorithm adapted by the framework from Section 3. First, for each input vector we define the initial discretization $\boldsymbol{\nu}_i = (\nu_{i,0}, \ldots, \nu_{i,D})^T \in \mathbb{R}^{D+1}$, e.g. as before by setting $\nu_{i,j} = j(\operatorname{Max}_i - \operatorname{Min}_i) + \operatorname{Min}_i$ where Min_i and Max_i is the minimal and the maximal value of the i-the feature and D is the maximal number of bins per feature. Let $\boldsymbol{\nu} = (\boldsymbol{\nu}_1^T, \ldots, \boldsymbol{\nu}_n^T)^T \in \mathbb{R}^{n(D+1)}$ be the concatenation of initial discretizations for all features. The SVM objective function

$$F_{\text{pwl}}^{\text{svm}}(\boldsymbol{w},\boldsymbol{\theta},\lambda) = \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max\left\{0, 1 - y_i f_{\text{pwl}}(\boldsymbol{x}^i, \boldsymbol{w}, \boldsymbol{\theta})\right\}$$

is then combined with the L_1 -norm approximation of the function

$$e_n(\boldsymbol{u}) = \sum_{i=1}^n \sum_{j=1}^{D-1} \llbracket u_{i,j} \neq \frac{1}{2} (u_{i,j-1} + u_{i,j+1}) \rrbracket.$$

The function $e_n(u)$ indicates the total number of weights that can be expressed as the average of its neighbors. Analogously to the general formulation (14), we obtain the following convex program

$$\boldsymbol{u}^{*} = \operatorname*{argmin}_{\boldsymbol{u} \in \mathbb{R}^{n(D+1)}} \left[F_{\mathrm{pwl}}^{\mathrm{svm}}(\boldsymbol{u}, \boldsymbol{\nu}, \lambda) + \gamma \sum_{i=1}^{n} \sum_{j=1}^{D-1} \left| u_{i,j} - \frac{1}{2} u_{i,j-1} - \frac{1}{2} u_{i,j+1} \right| \right].$$
(26)

The constant $\lambda \in \mathbb{R}_+$ controls the empirical error as in the standard SVM. The constant $\gamma \in \mathbb{R}_+$ controls the number of emerging bins, and thus the complexity (or smoothness) of the decision function. The uncompressed parameters u^* can be converted to the compressed ones (w^*, θ^*) by the projection methods from Section 4 which are applied to each pair $(u_i^*, \nu_i), i \in \{1, \ldots, n\}$, separately.

6 Limitations of the convex relaxation

In this section we discuss the limitations of the proposed convex relaxation. We concentrate on the PWC learning which leads to solving a convex minimization problem (8). The objective function (8) is a weighted sum of the original task objective $F_{\text{pwc}}(\boldsymbol{v}, \boldsymbol{\nu})$ and a regularization term $\sum_{i=1}^{D-1} |v_i - v_{i+1}|$ that was introduced to control the number of different neighboring weights. However, the regularization term can also have an undesired influence on the solution as shown below.

It is possible to factorize every candidate solution $\boldsymbol{v} \in \mathbb{R}^D$ into a monotonic subsequences whose endpoints are local extrema of the sequence (v_1, v_2, \ldots, v_D) . The value of the regularization term is given only by the weights at these endpoints. Indeed, let $v_k, v_{k+1}, \ldots, v_{k+l}$ be one of the monotonic subsequences and let it be an increasing one, that is, $v_k \leq v_{k+1} \leq \ldots \leq v_{k+l}$. In this case, we have

$$\sum_{i=k}^{k+l-1} |v_i - v_{i+1}| = \sum_{i=k}^{k+l-1} (v_{i+1} - v_i) = v_{k+1} - v_k$$

The same argument holds for decreasing subsequences, $v_k \ge v_{k+1} \ge \ldots \ge v_{k+l}$, in which case $\sum_{i=k+1}^{k+l} |v_{i-1} - v_i| = v_k - v_{k+l}$. This observation has the following two implications. First, the values of the intermediate weights, that is those between the local extrema, have no influence on the regularization term. Second, the only way to decrease the value of the regularization term is to either decrease a local maximum or to increase a local minimum. Subsequently the regularization term tends to push the local extrema towards each other and to make the resulting sequence flat.

Estimation of the PWC histograms by the ML method is an example application in which the described behavior is not favorable. In principle, we can use the same method as for estimation of the PWL histograms which was described in Section 7.2. The only difference is that for PWC histograms we use the regularization term $\gamma \sum_{i=1}^{D-1} |v_i - v_{i+1}|$ and we have to adjust the normalization constraint (24b) correspondingly. We applied the method to estimate the PWC histogram from 10⁶ examples randomly drawn from a known bimodal distribution (the high number was used to get rid of the estimation error). We estimated the PWC histogram with the regularization constant set to $\gamma \in \{0.1, 1, 10, 100\}$. The effect of increasing γ is illustrated in Figure 3. It is seen that the PWC regularization term favors uniform distributions which may not be desirable. This problem can be addressed by using the PWL histograms instead as shown in Section 7.2. In other applications, like learning linear classifiers on top of PWC histograms, the preference of uniform weights can be beneficial as shown in Section 7.1.

7 Experiments

This section provides empirical evaluation of the algorithms proposed in Section 5. Section 7.1 describes learning of malware detector representing the net-



Fig. 3 The PWC histogram learned by the ML method. The number of bins is controlled by the L1-regularization term $\gamma \sum_{i=1}^{D-1} |v_i - v_{i+1}|$. Each figure shows a histogram (blue) learned with a different setting of the regularization constant γ and the ground truth distribution (black) from which the training examples were drawn. As the regularization constant increases the undesired flattening effect of the PWC approximation becomes apparent.

work communication by PWC histograms. Section 7.2 evaluates the proposed PWL density estimator on synthetic data. Section 7.3 evaluates the proposed algorithm for learning PWL data embedding on classification benchmarks selected from the UCI repository.

7.1 Malware detection by classification of histograms

The proposed approach was applied in the network security domain to classify unseen malware samples from HTTP traffic. The data was obtained from several months (January - July 2015) of real network traffic of companies of various sizes in form of proxy log records. The logs contain anonymized HTTP/HTTPS web requests, where one request represents one connection defined as a group of packets from a single user and source port with a single server IP address, port, and protocol. Each request consists of the following fields: user name, source IP address, destination IP address, source port, destination port, protocol, number of bytes transferred from client to server and from server to client, flow duration, timestamp, user agent, URL, referer, MIME-Type, and HTTP status. The most informative field is the URL, which can be decomposed further into several parts, e.g. protocol, hostname, path, file name, or query.

Features applied on URL, path, query, file name
length; digit ratio
lower/upper case ratio; ratio of digits
vowel changes ratio
ratio of a character with max occurrence
has a special character
max length of consonant/vowel/digit stream
number of non-base64 characters
has repetition of parameters
Other Features
number of bytes from client to server
number of bytes from server to client
length of referer/file extension
number of parameters in query
number of '/' in path/query/referer

Table 1 List of selected connection-based features extracted from HTTP/HTTPS traffic.

We grouped all connections into bags, where one bag contains all connections of the same user going to the same domain. We the extracted 115 feature values for each connection (see Table 1), computed a histogram representation of each bag and used the histograms as input to a linear two-class classifier (21) as described in Bartos and Sofka (2015).

We compare two methods learning the classifier parameters:

- 1. The linear SVM using histograms with equidistantly spaced bins. The number of bins per feature varied from $\{8, 12, \ldots, 256\}$.
- 2. The proposed algorithm learning non-equidistant bins from examples. The uncompressed weights \boldsymbol{u}^* are obtained by solving (22) with the initial discretisation $\boldsymbol{\nu}$ set to split each feature equidistantly to D = 256 bins. The constant γ , which controls the number of bins, varied from 10^{-1} to 10^{-6} . The compressed weights $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ were obtained by the rounding procedure (15). Finally, the linear SVM was re-trained on the learned bins $\boldsymbol{\theta}^*$.

The optimal value of the SVM constant γ used by both compared methods was selected from $\{10^{-1}, \ldots, 10^{-5}\}$ based on the validation error.

The data consists of 7,028 positive (malware) and 44,338 negative (legitimate) samples. The positive samples are from 32 classes representing 32 different malware types. The samples were split into training, validation and testing set by a ratio 3/1/1. It is ensured that the same malware class never appears simultaneously in the training, validation and test sets. We report the average performance and its standard deviation computed over five test splits.

We analyzed the effect of the number of bins on the precision and recall of the linear SVM classifier. Figure 4 shows precision recall curves (PRCs) for the representation with different number of equidistant bins. Figure 5 shows PRCs of the representation with different number of non-equidistant bins positions of which are learned by the proposed method. It is seen that the representation with non-equidistant bins achieves higher precision using substantially smaller



Fig. 4 Precision recall curves for histogram representation with different number of equidistant bins. The performance increases with the number of bins, however higher number of bins show comparable results.



Fig. 5 Precision recall curves for histogram representation with non-equidistant bins learned by the proposed method. The performance is increased when compared to Figure 4, and with smaller number of bins.





Fig. 6 SVM weights learned with three different algorithms: a) linear SVM with 512 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.

Fig. 7 Recall is influenced by the complexity of the baseline histogram representation as well as of the representation learned with the proposed approach. However, the proposed optimization achieves higher recall with significantly less number of bins.

number of bins. In Figure 7 we show the recall as the function of the number of bins in order to emphasize the difference between the equidistant and learned non-equidistant representation. The precision was set at 95%, a common operating point of the detector used in real-life deployment. The figures are numerically summarized in Table 2.

Figure 6 illustrates the weights \mathbf{w} of the linear SVM classifier trained with three different representations. First, we used a histograms with 256 equidistant bins (black line), resulting in a large complexity of weights. Second, we learned the weights and bins simultaneously according to the proposed algorithm (red line), which significantly decreased the number of bins without sacrificing the efficacy. Finally, we applied the second step to define new bins

Vojtech Franc, Ondrej Fikar, Karel Bartos, Michal Sofka

Equidistant bins			Learned soft bins		Learned rounded bins			
bins per	В	recall at 95	γ	recall at 95	bins per	В	recall at 95	
feature		[%]		[%]	feature		[%]	
256	$58,\!880$	53.5(25.4)	$5 \cdot 10^{7}$	58.2(24.4)	58	13,316	58.9(23.6)	
128	29,440	51.0(27.9)	$1 \cdot 10^{6}$	56.4 (23.9)	40	9,196	58.3(22.9)	
64	14,720	51.2(26.5)	$5 \cdot 10^{6}$	56.4(22.6)	20	4,634	55.0(20.6)	
32	7,360	50.3(26.3)	$1 \cdot 10^{5}$	56.2(24.3)	13	2,991	54.5(22.2)	
16	$3,\!680$	46,7(26.9)	$5 \cdot 10^{5}$	54.6 (25.4)	3	741	51.1 (25.7)	
8	1,840	45.6 (28.5)	$1 \cdot 10^{4}$	51.2 (22.5)	2	510	50.0 (27.5)	

Table 2 Performance comparison of a linear SVM classifier trained from a histogram representation with equidistant bins with the two proposed methods: learned soft bins (when the bins and SVM weights are learned simultaneously) and learned rounded bins (retraining new SVM weights once the bins are learned from the samples). All approaches have comparable recall, but the proposed algorithms significantly reduced the number of bins.

by rounding the uncompressed solution and then re-trained a new linear SVM classifier on the top of it (blue line). It means that the discretization for the second and third method is similar, but the third method retrained the classifier and boosted the weights to further maximize the separability of the positive and negative samples.

7.2 Non-parametric distribution estimation

We evaluated the algorithm finding PWL approximation of an unknown distribution described in Section 5.2. The samples were drawn from a mixture of two Gaussians: $p(x) = 0.4 \cdot \mathcal{N}(x; -2, 1) + 0.6 \cdot \mathcal{N}(x; 2, 0.5)$ with mean and standard deviation (-2, 1) and (2, 0.5), respectively. We compared three methods:

- 1. The proposed algorithm estimating non-equdistant PWL histogram. The uncompressed weights \boldsymbol{u}^* were obtained by solving (24). The initial D = 100 bins $\boldsymbol{\nu}$ were placed equidistantly between the minimal and the maximal value in the training set. The optimal value of the constant γ was selected from $\{10, \ldots, 10000\}$ based on the log-likelihood evaluated on the validation set. The compressed parameters $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ of the PWL histogram (23) were computed from \boldsymbol{u}^* by the rounding procedure (20) with the precision parameter $\varepsilon = 0.001$.
- 2. The PWL histogram with bin edges $\boldsymbol{\theta}$ placed equidistantly between the minimal and the maximal value in the training set. The weights \boldsymbol{w} were found by maximizing the likelihood function which is equivalent to solving (24) with $\gamma = 0$. The optimal number of bins was selected from $\{5, 10, \ldots, 100\}$ based on the log-likelihood evaluated on the validation set.
- 3. The standard PWC histogram with equidistant bins whose number was selected from $\{5, 10, \ldots, 100\}$ based on the log-likelihood evaluated on the validation set. The bin heights were found by the ML method.

We used the distribution p(x) to generate training and validation set the size of which varied from 100 to 10000. For each method we recorded the optimal number of bins and the KL-divergence between the estimated and the

ground distribution p(x). The results are averages and standard deviations computed over ten generated data sets.

Figure 8(a) shows the KL-divergence and Figure 8(b) the number of bins as a function of the training set size. As expected, the equidistant PWC histogram provides the least precise (high KL divergence) and the most complex (high number of bins) model. We also see that PWL model with equidistantly-spaced bins provides the same accurate model as the model with non-equidistant bins learned from example, however, the compactness (the number of bins) of the non-equidistant model is consistently smaller.

Figure 9 shows examples of PWC historams and PWL histograms with learned non-equdistant bins. It is seen that the non-equdistant PWL histogram can closely approximate the ground truth model even from small training sets.



Fig. 8 Figures compare the proposed method learning non-equidistant PWL histograms (black), the equidistant PWL histogram (green) and the equidistant PWC histogram (blue). Figure (a) depicts the KL-divergence between the ground truth and the estimated model as a function of the training set size. Figure (b) shows the optimal number of bins as the function of training set size.

7.3 PWL embedding for non-linear classification

In this section we evaluate the algorithm for learning the PWL data embedding proposed in Section 5.3. We learned a two-class classifier $h(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}) =$ $\operatorname{sign}(f_{\mathrm{pwl}}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta}))$ with the discriminant function $f_{\mathrm{pwl}}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{\theta})$ defined by (25) for a set of classification problems selected from the UCI repository (Lichman, 2013) which are summarized in Table 3. We evaluated three methods:

1. The proposed algorithm learning simultaneously $\boldsymbol{\theta}$ and \boldsymbol{w} . The uncompressed parameters \boldsymbol{u}^* were found by solving (26) with the initial discretization $\boldsymbol{\nu}$ equidistantly splitting each feature to D = 100 bins. The compressed parameters $(\boldsymbol{w}^*, \boldsymbol{\theta}^*)$ were computed from $(\boldsymbol{u}^*, \boldsymbol{\nu})$ by the rounding procedure (20) with the precision parameter $\varepsilon = 0.1$. Finally, a linear



Fig. 9 Figures show the ground truth p.d.f. (red), the learned PWL histogram (black) and the PWC histogram with equidistant bins (blue) estimated from training sets of different sizes. The number of bins of a particular histogram is shown in brackets. The black vertical lines denote the learned bin edges.

SVM was re-trained on the learned bins θ^* . The constant γ , which controls the number of bins, was varied from 0.1 to 0.0001.

- 2. The parameters \boldsymbol{w} were trained by the linear SVM on top of equdistantly constructed bins $\boldsymbol{\theta}$. The number of bins per feature varied from 5 to 20.
- 3. Method of (Pele et al., 2013) which was shown to outperform the non-linear SVM with many state-of-the-art kernels and data embeddings. Namely, we re-implemented the "PL1 algorithm" applying the PWL embedding on individual features as we do. The non-equidistant bins were found for each feature independently by constructing edges as the mid-points between the cluster centers obtained from the k-means algorithm. The number of bins was varied from 3 to 20.

The optimal value of the SVM constant λ used by all three methods was selected from $\{0.1, \ldots, 0.00001\}$ based on the validation error.

We used the same evaluation protocol as in (Pele et al., 2013). Each data set was ten times randomly split into training, validation and test part in the ratio 60/20/20. The reported results are averages and the standard deviations computed on the test part over the ten splits.

Learning data discretization via convex optimization

name	num. of	num. of	Pele et. al	2013	Equdist. bins		Learned bins	
	examples	features	acc %	bins	acc %	bins	acc %	bins
eyestate	14,980	15	64.6 ± 7.1	20.0	60.5 ± 5.1	21.0	72.9±4.0	13.3
ionosphere	351	34	92.1 ± 1.9	3.0	90.9 ± 2.3	6.0	92.2±0.2	8.4
magic	19,020	11	85.4 ± 0.3	20.0	$85.4 {\pm} 0.3$	21.0	86.0±0.4	11.7
miniboo	130,065	50	91.0 ± 0.2	3.0	$84.4 {\pm} 0.6$	21.0	$92.2{\pm}0.2$	5.7
musk	6,598	167	98.8 ± 0.3	10.0	$98.4 {\pm} 0.4$	11.0	99.0±0.2	20.2
skin	245,057	4	96.5 ± 0.2	20.0	$96.6 {\pm} 0.1$	21.0	$97.3{\pm}0.1$	19.3
sonar	208	60	$84.1{\pm}5.8$	5.0	79.3 ± 5.8	6.0	82.0±3.7	3.7
spectf	267	44	$79.4{\pm}4.9$	10.0	79.2 ± 0.0	6.0	79.3 ± 3.2	4.6
transf	748	5	76.4 ± 0.7	3.0	$77.0{\pm}1.3$	11.0	76.3 ± 0.6	3.0
wilt	4,889	6	97.0 ± 0.6	20.0	$94.8 {\pm} 0.3$	21.0	98.6±0.2	9.7

Table 3 A summary of two-class classification problems selected from the UCI repository (Lichman, 2013) and used to evaluate the linear embedding algorithms. Further, the "skin" database originates from Bhatt and Dhall (2010), "wilt" from Johnson et al. (2013) and "transf" from Yeh et al. (2008). The right part of the table summarizes the classification accuracy for the best choice of bins and the corresponding number of bins. The reported values are means and standard deviations computed over 10 random splits.

Figure 10 shows the test classification accuracy of the compared methods as a function of the number of bins. The accuracy for the best choice of bins and the corresponding number of bins is summarized in Table 3. The baseline PWL embedding with equidistant bins provides slightly but consistently worse accuracy compared to the other two methods which learn the non-equidistant bins. The proposed method yields comparable or slightly better results than the approach of (Pele et al., 2013) both in terms of the classification accuracy and the complexity of the embedding (number of bins). However, the proposed method relies solely on solving convex optimization problems unlike the method of (Pele et al., 2013) which involves a highly non-convex clustering problem.

7.4 The computational time

In this section we provide empirical estimate of the computational time required when learning discretization by the proposed framework. As a benchmark we use the task of learning the PWL embedding for non-linear classification described in Section 7.3. In this case learning leads to a convex optimization problem (26) which has two hyper-parameters. First, the parameter λ controls the quadratic regularization similar to the standard SVM formulation. Second, the additional parameter γ which implicitly controls the number of bins of the resulting feature discretization. The time required to solve the convex problem thus depends on the hyper-parameters λ and γ and on the size of the optimized problem specified by the number of examples m and the number of features n. We empirically measured the dependency of the computational time on the four variables as described below.

Besides the four variables, the runtime obviously depends on a particular optimization solver. There is a large number of optimization methods applicable to the convex problem (26). For example, the problem (26) can be expressed as an equivalent quadratic program (QP) and solved by any QP solver. In this work we used the Optimized Cutting Plane Algorithm for Large-Scale Risk Minimization (OCA) (Franc and Sonneburg, 2009). The OCA is a general purpose and easy-to-implement solver for minimization of convex functions containing a quadratic regularization term like the problem (26). The empirical study presented below uses this particular solver which is sufficiently efficient for the problem sizes considered in our experiments. Other solvers might be even more efficient.

We used a Matlab implementation of the OCA solver. The experiments were performed on a Linux machine with 32 CPUs (AMD Opteron 1.4GHz) and 256 GB RAM. We stopped the OCA solver when the objective value reached a level not more than 1% above the optimal value. Such precision may be exaggeratedly high for practical purposes but it removes the influence of the optimization error on the resulting accuracy.

Figure 11 shows the average runtime required by the OCA solver as a function of λ , γ , the number of examples m and the number of features n. The runtime is measured on a subset of classification problems listed in Table 3. In case of λ and γ we use problems with more than 1,000 examples (6 problems in total). The dependency on the number of examples and the number of features is measured on "miniboo" and "musk" dataset, respectively.

We found that the runtime scales gracefully with respect to the quadratic regularization parameter λ , specifically, it grows approximately as $\mathcal{O}(\lambda^{-0.8})$. On the other hand, the runtime grows much faster, approximately $\mathcal{O}(e^{20\sqrt{\gamma}})$, with increasing γ controlling the number of bins. It means that the lower number of bins the higher computational time is needed. A low number of bins is enforced by increasing the weight of the L_1 -regularization term in the objective function, which brings the task closer to a linear program. The dominant linear term impairs the regularization effect of the quadratic term which causes the "zig-zag" behavior of the cutting plane solver leading to a higher number of iterations. The dependency of the running time on the number of training examples is linear which is consistent with the theoretical upper bound on the number of iterations proved in Franc and Sonneburg (2009). Finally, the dependency on the number of features is approximately quadratic.

In absolute numbers, the longest time, around 160 minutes, was required for training a single model on "miniboo" dataset (130,065 examples, 50 features) with the highest value of $\lambda = 0.1$.

8 Conclusions

We proposed a generic framework which allows to modify a wide class of convex learning algorithms so that they can learn parameters of the piece-wise constant (PWC) and the piece-wise linear (PWL) functions from examples. The learning objective of the original algorithm is augmented by a convex term which enforces compact bins to emerge from an initial fine discretization. In contrast to existing methods, the proposed approach learns the discretization and the parameters of the decision function simultaneously. In addition, learning is converted to a convex problem which is solvable efficiently by global methods. We instantiated the proposed framework for three problems, namely, learning PWC histogram representation of sequential data, estimation of the PWL probability density function and learning PWL data embedding for nonlinear classification. The proposed algorithms were evaluated on synthetic data and standard public benchmarks and applied to malware detection in network traffic data. It was demonstrated that the proposed convex algorithms yield models with fewer number of parameters with comparable or better accuracy than the existing methods. The main disadvantage of the proposed method, when compared to heuristic local methods like Pele et al. (2013), is a higher computational time required to solve the convex problem.

Acknowledgements VF was supported by Czech Science Foundation grant 16-05872S. OF was supported by the internal CTU funding SGS17/185/OHK3/3T/13.

References

- Bartos, K. and Sofka, M. (2015). Robust representation for domain adaptation
- in network security. In *In proc. of ECML/PKDD*, volume 3, pages 116–132.
- Bhatt, R. and Dhall, A. (2010). Skin segmentation dataset. UCI Machine Learning Repository.
- Boullé, M. (2006). Modl: A bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165.
- Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.
- Candes, E., Romberg, J., and Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8).
- Candes, E. and Tao, T. (2005). Decoding by linear programming. IEEE Transactions on Infromation Theory, 51(12):4203–4215.
- Chapelle, O., Haffner, P., and Vapnik, V. N. (1999). Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Net*works, 10(5):1055–1064.
- Cios, K., Pedrycz, W., Swiniarski, R., and Kurgan, L. (2007). *Data Mining:* A Knowledge Discovery Approach. Springer US.
- Dalal, N. and Triggs, B. (2005). Histogram of oriented gradients for human detection. In Proc. of Computer Vision and Pattern Recognition, volume 1, pages 886–893.
- Donoho, D. (2006). Compressed sensing. IEEE Transactions on Infromation Theory, 52(4):1289–1306.
- Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In Proc. of International Conference on Machine Learning, pages 194–202. Morgan Kaufmann.

- Elomaa, T. and Rousu, J. (1999). General and efficient multisplitting of numerical attributes. *Machine Learning*, 36(3):201–244.
- Fayyad, U. M. and Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8(1):87–102.
- Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In Proc. of International Joint Conference on Artificial Intelligence, pages 1022–1029.
- Franc, V. and Sonneburg, S. (2009). Optimized cutting plane algorithm for large-scale risk minimization. *Journal of Machine Learning Research*, 10:2157–2232.
- Friedman, N. and Goldszmidt, M. (1996). Discretizing continuous attributes while learning bayesian networks. In Proc. of Internatinal Conference on Machine Learning, pages 157–165.
- García, S., Luengo, J., Sáez, J. A., López, V., and Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750.
- Hue, C. and Boullé, M. (2007). A new probabilistic approach in rank regression with optimal bayesian partitioning. *Journal of Machine Learning Research*, 8:2727–2754.
- Johnson, B. A., Tateishi, R., and Hoan, N. T. (2013). A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International journal of remote sensing*, 34(20):6969– 6982.
- Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In Proc. of the Tenth National Conference on Artificial Intelligence, AAAI'92, pages 123–128.
- Lichman, M. (2013). UCI machine learning repository.
- Liu, H., Hussain, F., Tan, C. L., and Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423.
- Pele, O., Taskar, B., Globerson, A., and Werman, M. (2013). The pairwise piecewise-linear embedding for efficient non-linear classification. In Proc. of the International Conference on Machine Learning, pages 205–213.
- Rao, C. (2005). Handbook of Statistics: Data Mining and Data Visualization. Handbook of Statistics. Elsevier Science.
- Silverman, B. (1986). Density Estimation for Statistics and Data Analysis. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis.
- Yang, Y. and Webb, G. I. (2008). Discretization for naive-bayes learning: managingdiscretization bias and variance. *Machine Learning*, 74(1):39–74.
- Yeh, I.-C., Yang, K.-J., and Ting, T.-M. (2008). Knowledge discovery on rfm model using bernoulli sequence. *Expert Systems with Applications*, 36(3):5866–5871.



Fig. 10 Classification accuracy (mean and std computed over ten splits) of the linear classifier using PWL data embedding evaluated on problems selected from UCI repository. The accuracy is shown as a function of the average number of bins used to discretize each feature. Results are shown for the baseline using equidistantly placed bins (blue), the method of (Pele et al., 2013) finding non-equidistant bins by the k-means algorithm (green) and the proposed methods learning the non-equidistant bins by a convex programming (black).



Fig. 11 Figures report the computational time required by the OCA solver to find a precise solution (within 1% from the optimal value) of the convex problem (26). The runtime is shown as a function of the hyper-parameter λ (upper left), the hyper-parameter γ (upper right), the number of examples m (bottom left) and the number of features n (bottom right). The runtime for λ and γ is measured on a subset of UCI datasets with more than 1,000 examples (cf. Table 3). The runtime for the number of examples m and the number of features n is measured on "miniboo" and "musk" dataset, respectively. The figures show average time and standard deviations computed over 10 runs of the solver. When measuring γ , the value of λ was fixed to 0.1 (middle of the range). Similarly, when measuring λ , the value of γ was fixed to 0.01 (middle of the range). The dependency for m and n was measured when the hyper-parameters were fixed to $\gamma = 0.01$ and $\lambda = 0.1$.