Smooth Maximum Based Algorithms for Classification, Regression, and Collaborative Filtering

G. Takács

Széchenyi István University, Győr, Hungary

Abstract: Unbalanced classification problems are quite common in the practice of machine learning. Unbalancedness means that the distribution of the class labels is far from uniform. Convex polyhedron classifiers are special binary classifiers that fit well to unbalanced problems. In this paper I propose novel and computationally efficient algorithms for training convex polyhedron classifiers. The proposed algorithms are based on the smooth approximation of the maximum function. I also give the analogous variant of the approach for regression and collaborative filtering. Finally, I demonstrate the usefulness of the approach via experiments on artificial and real datasets.

Keywords: classification, regression, collaborative filtering, convex polyhedron, smooth maximum function

1. Introduction

In this paper *machine learning* is considered as discovering the relationship between the features of a phenomenon, based on a dataset that was collected by observing the phenomenon. Classification, regression, and collaborative filtering are three important special cases of machine learning.

In the problem of *classification* the phenomenon is modeled by a random pair $(\mathbf{X},Y)$, where

- $\mathbf{X}$ taking values from $\mathbb{R}^d$ is called *input*, and
- Y taking values from $\mathcal{C} = \{c_1,\ldots,c_M\}, M \geq 2$ is called *label*. If $M = 2$, then the problem is termed *binary classification*, otherwise it is termed *multiclass classification*.

The goal is to predict Y from $\mathbf{X}$ with a function[1] $g : \mathbb{R}^d \mapsto \mathcal{C}$ called *classifier* such that the probability of error

$$L(g) = \mathbf{P}\{g(\mathbf{X}) \neq Y\}$$

[1] Functions are always assumed to be measurable in this paper. Otherwise the function of a random variable would not necessarily be a random variable.

is minimal.

Typically, the distribution of $(\mathbf{X},Y)$ is unknown, therefore the minimal error probability and the optimal classifier are unknown too. We only have a finite sequence of corresponding input–label pairs from the past

$$\mathbf{T} = ((\mathbf{X}_1,Y_1),\ldots,(\mathbf{X}_n,Y_n)),$$

called training set. It is assumed that these pairs were drawn independently from the unknown distribution of $(\mathbf{X},Y)$, and also that $(\mathbf{X},Y)$ and $\mathbf{T}$ are independent. In practice we usually observe only one realization of $\mathbf{T}$ denoted by $\mathbf{t} = ((\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n))$. This is our data at hand that we have to live with. The task is to *estimate* the optimal classifier on the basis of $\mathbf{T}$.

In the problem of *regression* the phenomenon is a random pair $(\mathbf{X},Y)$, where

- $\mathbf{X}$ taking values from $\mathbb{R}^d$ is called *input*, and
- Y taking values from $\mathbb{R}$ is called *target*.

The goal is to predict Y from $\mathbf{X}$ with a function $g : \mathbb{R}^d \mapsto \mathbb{R}$ called *predictor* such that the mean squared error

$$L(g) = \mathbf{E}\{(g(\mathbf{X}) - Y)^2\}$$

is minimal. It is true again that typically we have no information about the distribution of $(\mathbf{X},Y)$. The task is to estimate the optimal predictor based on an independent and identically distributed sample.

In *collaborative filtering*, the phenomenon is a random triplet (U,I,R), where

- U taking values from $\{1,\ldots,N_U\}$ is called the *user identifier*,
- I taking values from $\{1,\ldots,N_I\}$ is called the *item identifier*, and
- R taking values from $\{v_1,\ldots,v_M\} \subset \mathbb{R}$ is called the *rating value*.

A realization of (U,I,R) denoted by (u,i,r) means that the u-th user rated the i-th item with value r. The goal is to predict R from $(U,\ I)$ with a function $g : \{1,\ldots,N_U\} \times \{1,\ldots,N_I\} \mapsto \{v_1,\ldots,v_M\}$ such that mean squared error

$$L(g) = \mathbf{E}\{(g(U,I) - R)^2\}$$

is minimal.

Collaborative filtering can be viewed as a special case of regression. However, classical regression techniques are not suitable for solving collaborative filtering problems, because of the unique characteristics of the input variables.

Denote the random training set by $\mathbf{T} = ((U_1,I_1,R_1),\ldots,(U_n,I_n,R_n))$, and its realization by $\mathbf{t} = ((u_1,i_1,r_1),\ldots,(u_n,i_n,r_n))$. Denote the set of user–item pairs appearing in the training set by $\mathcal{T} = \{u,i : \exists k : u_k = u,i_k = i\}$.

In real life, if a user has rated an item, then it is unlikely that he/she will rate the same item again. Therefore it is unrealistic to assume that the elements of the training set are independent. A more reasonable assumption is

$$\mathbf{P}\{U_k = u_k,I_k = i_k,R_k = r_k\} = \\ \mathbf{P}\{U = u_k,I = i_k,R = r_k | \cap_{l=1}^{k-1} (U \neq u_l,I \neq i_l)\},$$

which means that the training set is generated by a "sampling without replacement" procedure.

If this assumption holds, then the training data can be represented as a partially specified matrix $\mathbf{R} \in \mathbb{R}^{N_U \times N_I}$ called rating matrix, where the matrix elements are known in positions $(u,i) \in \mathcal{T}$, and unknown in positions $(u,i) \notin \mathcal{T}$. The value of the matrix $\mathbf{R}$ at position $(u,i) \in \mathcal{T}$, denoted by r_{ui}, stores the rating of user u for item i.

1.1. Convex polyhedron classification

Many interesting classification problems arising in practice are *unbalanced*, which means that the distribution of labels is far from uniform. For example, in the case of breast cancer screening most patients are (fortunately) healthy. This results that in the corresponding binary classification problem most training examples belong to the "healthy" class. Convex polyhedron classifiers are special nonlinear classifiers that fit well to unbalanced problems.

Let us consider an unbalanced binary classification problem with labels c_1 and c_2. Let us call c_1 the positive and c_2 the negative class, and assume that the class with higher probability is the negative class. A convex K-polyhedron (polyhedron) is the intersection of K half-spaces (any number of half-spaces).

A *convex polyhedron (K-polyhedron) classifier* is a function $g : \mathbb{R}^d \mapsto \{c_1,c_2\}$ such that $\{\mathbf{x} \in \mathbb{R}^d : g(\mathbf{x}) = c_1\}$ is a convex polyhedron (K-polyhedron). An equivalent definition is the following: A function $g : \mathbb{R}^d \mapsto \{c_1,c_2\}$ is called a convex K-polyhedron classifier, if it can be written as

$$\begin{aligned} g(\mathbf{x}) =& \mathrm{th}(\min\{\mathbf{w}_1^T\mathbf{x} + b_1,\ldots,\mathbf{w}_K^T\mathbf{x} + b_K\}) \\ =& \mathrm{th}(-\max\{-\mathbf{w}_1^T\mathbf{x} - b_1,\ldots,-\mathbf{w}_K^T\mathbf{x} - b_K\}), \end{aligned} \tag{1}$$

where $\mathbf{w}_1,\ldots,\mathbf{w}_K$ are called weight vectors, $b_1,\ldots,b_K$ are termed biases, and

$$\mathrm{th}(z) = \begin{cases} c_1 & \text{if } z \geq 0 \\ c_2 & \text{if } z < 0 \end{cases}$$

is the threshold function.

When classifying an input $\mathbf{x}$, we iterate over the weight vectors. If $\mathbf{w}_k^T\mathbf{x} + b_k < 0$ for any $k \in \{1,\ldots,K\}$, then the input can be classified as negative immediately. As a consequence, convex polyhedron classifiers tend to classify negative examples quickly. This property makes the approach particularly suitable for unbalanced problems.

Despite this appealing property, currently convex polyhedron classifiers are not frequently used in practice. The main reason for that is the lack of efficient and practical training algorithms. In the next section we will overview the small literature of the area. Then, I will propose novel algorithms that attempt to make the convex polyhedron classifier a practical tool.

2. Known methods

Probably the best known work that applied convex polyhedron classifiers for solving a practical problem is [10]. In this paper the authors propose the *maximal rejection* (MR) approach that can be applied for training convex polyhedron classifiers. The key idea of MR is defining the criterion function

$$\mathcal{M}(\mathbf{w}) = \frac{(\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_0)^2 + \mathbf{w}^T\mathbf{R}_1\mathbf{w} + \mathbf{w}^T\mathbf{R}_0\mathbf{w}}{\mathbf{w}^T\mathbf{R}_1\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}}, \tag{2}$$

where $\mathbf{m}_1$, $\mathbf{m}_0$, $\mathbf{R}_1$ and $\mathbf{R}_0$ are the empirical means and covariances of the classes in the training set, and λ is the regularization coefficient. The detailed derivation of this criterion function (without the regularization term $\lambda\mathbf{w}^T\mathbf{w}$) can be found in [10]. The main idea is to modify the criterion function of Fisher discriminant analysis [12] such that we allow more variance within class 0, if the variance within class 1 is smaller. If we introduce the notation $\mathbf{Q} = (\mathbf{m}_1 - \mathbf{m_0})(\mathbf{m}_1 - \mathbf{m_0})^T + \mathbf{R}_1 + \mathbf{R}_0$, then $\mathcal{M}$ can be written as

$$\mathcal{M}(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{Q}\mathbf{w}}{\mathbf{w}^T(\mathbf{R}_1 + \lambda\mathbf{I})\mathbf{w}},$$

where $\mathbf{I}$ is the $d \times d$ identity matrix.

It can be shown that the $\mathbf{w}$ that maximizes $\mathcal{M}$ is an eigenvector of $(\mathbf{R}_1 + \lambda\mathbf{I})^{-1}\mathbf{Q}$ corresponding to the largest eigenvalue. Note that the maximum is not unique, since $\mathcal{M}(\mathbf{w}) = \mathcal{M}(\alpha\mathbf{w})$ for every $\alpha \neq 0$.

The outline of MR training is the following[2]:

- For $k = 1,\ldots,K$:

[2]This variant is a bit more flexible than the original one.

– Set $\mathbf{w}_k$ to $\arg\max_{\mathbf{w}\in\mathbb{R}} \mathcal{M}(\mathbf{w})$.
– If $\sum_{i:y_i=1} \mathbf{w}_k^T\mathbf{x}_i / \sum_i y_i < \sum_{i:y_i=0} \mathbf{w}_k^T\mathbf{x}_i / \sum_i (1-y_i)$, then flip the sign of $\mathbf{w}_k$.
– Define $g_k(\mathbf{x})$ as $\mathrm{th}(\min\{\mathbf{w}_1^T\mathbf{x}+b_1,\ldots,\mathbf{w}_k^T\mathbf{x}+b_k\})$.
– Set b_k by minimizing $\sum_{i:y_i=1} I\{g_k(\mathbf{x}_i)\neq y_i\} + \beta\sum_{i:y_i=0} I\{g_k(\mathbf{x}_i)\neq y_i\}$.
– Exclude examples from the training set for which $g_k(\mathbf{x}_i)=0$.

The output of training is a convex K-polyhedron classifier. The parameter $\beta > 0$ expresses our willingness to tolerate false negative classifications (larger β results more false negatives and less false positives).

There also exist other known methods for training convex polyhedron classifiers, but they are less practical than MR. Some of the alternatives are the following:

- [17] tries to separate each negative example from the positive class individually with an adaptation of the multiclass support vector machine method [8]. The algorithm can be used only for small problems due to its large computational complexity.
- The training algorithm if the ID3 decision tree [18] can also be used for training convex polyhedron classifiers, if we introduce the following restrictions: all features have to be continuous or binary, and one of the partitions has to be labeled as negative after each split. Unfortunately, the modeling power of this approach is quite limited.
- In the literature of *probably approximately correct learning* (PAC learning) [22] one can find theoretical works related to convex polyhedron classification, for example [11, 14, 15, 23]. PAC is a formalism for determining how much data is needed for a given classification algorithm to achieve a given accuracy on a given fraction of test examples. Unfortunately, the convex polyhedron classification algorithms published in the PAC papers are not practical methods. They are instead tools for proving theorems about PAC-learnability.

3. Smooth maximum functions

One of the factors that make the training of convex K-polyhedron classifiers hard is the non-differentiable maximum function appearing in the definition formula. One possible way of handling the difficulty is approximating maximum taking with a smooth[3] function.

Let us start the discussion with a simple observation. Assume that we have K different real numbers $u_1,\ldots,u_K$, and a function $f:\mathbb{R}\mapsto\mathbb{R}$ with the following property:

$$\forall u\in\mathbb{R}: \lim_{\Delta\to\infty}\frac{f(u+\Delta)}{f(u)} = \infty.$$

[3]Infinitely many times differentiable.

Denote the largest number by $u_{\max} = \max\{u_1,\ldots,u_K\}$, and the smallest number by $u_{\min} = \min\{u_1,\ldots,u_K\}$. Let us apply f on the numbers and investigate the values $f(u_1),\ldots,f(u_K)$. If the difference between $u_{\max}$ and the other numbers is large enough, then the following approximation is admissible:

$$\frac{f(u_j)}{f(u_{\max})} = \frac{f(u_j)}{f(u_j + (u_{\max} - u_j))} \approx \begin{cases} 0 & \text{if } u_j \neq u_{\max}, \\ 1 & \text{if } u_j = u_{\max}. \end{cases} \tag{3}$$

It follows from (3) that

$$\frac{f(u_j)}{\sum_{k=1}^{K} f(u_k)} = \frac{f(u_j)/f(u_{\max})}{\sum_{k=1}^{K} f(u_k)/f(u_{\max})} \approx \begin{cases} 0 & \text{if } u_j \neq u_{\max}, \\ 1 & \text{if } u_j = u_{\max}. \end{cases} \tag{4}$$

If f is monotonically increasing and smooth, then based on (4) it is possible to define smooth approximations for the maximum function:

$$\begin{aligned} &\text{A)}\ \max\{u_1,\ldots,u_K\} \approx f^{-1}\left(\sum_{k=1}^{K} f(u_k)\right), \\ &\text{B)}\ \max\{u_1,\ldots,u_K\} \approx f^{-1}\left(\frac{1}{K}\sum_{k=1}^{K} f(u_k)\right), \\ &\text{C)}\ \max\{u_1,\ldots,u_K\} \approx \sum_{j=1}^{K} \frac{f(u_j)}{\sum_{k=1}^{K} f(u_k)} u_j. \end{aligned} \tag{5}$$

Schemes A and B are similar: the only difference between them is the $\frac{1}{K}$ factor appearing in B. An advantage of A over B is that it approximates the max function better, if the difference between $u_{\max}$ and the other numbers is large. An advantage of B over A is that its result is always between $u_{\min}$ and $u_{\max}$. Scheme C is an interesting one: it calculates the answer by assigning a weight to each variable, and it does not need the inverse of f. It is also true for C that the output is always between $u_{\min}$ and $u_{\max}$.

The most natural choice for f is the exponential function $f(u) = \exp(\alpha u)$,$\alpha > 0$. The power function $f(u) = u^{\alpha}$,$\alpha > 1$ is also suitable in the nonnegative domain. With the given approximation schemes and f functions we can define 6 different smooth maximum

functions:

$$\begin{aligned}
\mathrm{smax}_{A1}(\mathbf{u}) &= \frac{1}{\alpha}\ln\left(\sum_{k=1}^{K}\exp(\alpha u_k)\right)\\
\mathrm{smax}_{A2}(\mathbf{u}) &= \left(\sum_{k=1}^{K}u_k^{\alpha}\right)^{1/\alpha}\\
\mathrm{smax}_{B1}(\mathbf{u}) &= \frac{1}{\alpha}\ln\left(\frac{1}{K}\sum_{k=1}^{K}\exp(\alpha u_k)\right)\\
\mathrm{smax}_{B2}(\mathbf{u}) &= \left(\frac{1}{K}\sum_{k=1}^{K}u_k^{\alpha}\right)^{1/\alpha}\\
\mathrm{smax}_{C1}(\mathbf{u}) &= \sum_{j=1}^{K}\frac{\exp(\alpha u_j)}{\sum_{k=1}^{K}\exp(\alpha u_k)}u_j\\
\mathrm{smax}_{C2}(\mathbf{u}) &= \sum_{j=1}^{K}\frac{u_j^{\alpha}}{\sum_{k=1}^{K}u_k^{\alpha}}u_j
\end{aligned} \tag{6}$$

where $\mathbf{u} = [u_1,\ldots,u_K]$ denotes the vector containing all numbers. Parameter α can be used to control the "degree of smoothness" (larger α results better approximation, but less smooth functions). Note that smax_{A1} and smax_{B1} differ only in a constant, and smax_{A2}, smax_{B2}, smax_{C2} are admissible only if $u_1,\ldots,u_K$ are all non-negative [4]. The surface plot of the maximum function in 2 dimensions can be seen in Figure (1). The presented smooth maximum functions are depicted in Figure (2) and their difference from $\max$ in Figure (3).

Two simple properties of the maximum function are interchangeability with constant addition and non-negative constant multiplication:

$$\begin{aligned}
\max\{u_1 + C,\ldots,u_K + C\} &= \max\{u_1,\ldots,u_K\} + C,\\
\max\{Cu_K,\ldots,Cu_K\} &= C\max\{u_1,\ldots,u_K\},
\end{aligned}$$

where C is an arbitrary constant in the first case and a non-negative constant in the second case. Interestingly, for 5 of the given smooth maximum functions exactly *one* of these properties is true (smax_{A1}, smax_{B1} and smax_{C1} have the first, smax_{B2} and smax_{C2} have the second property).

[4] $\mathrm{smax}_{C2}(\mathbf{0})$ can be defined as zero.

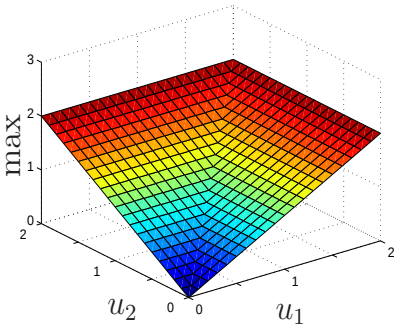


Figure 1: The maximum function in 2 dimensions.

Let us introduce the following abbreviation ($j = 1, \ldots, K$):

$$p_j = \frac{f(u_j)}{\sum_{k=1}^{K} f(u_k)}.$$

The quantity p_j can be interpreted as a "measure of dominance" of the j-th number over the others. If $f(u) = \exp(\alpha u)$, then $p_j = \frac{\exp(\alpha u_j)}{\sum_{k=1}^{K} \exp(\alpha u_k)}$. If $f(u) = u^\alpha$, then $p_j = \frac{u_j^\alpha}{\sum_{k=1}^{K} u_k^\alpha}$.
The partial derivatives of the proposed smooth maximum functions are ($j = 1, \ldots, K$):

$$\begin{aligned}
\mathrm{smax}'_{j,A1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{A1}}{\partial u_j}(\mathbf{u}) = p_j, \\
\mathrm{smax}'_{j,A2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{A2}}{\partial u_j}(\mathbf{u}) = p_j \frac{s}{u_j}, \\
\mathrm{smax}'_{j,B1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{B1}}{\partial u_j}(\mathbf{u}) = p_j, \\
\mathrm{smax}'_{j,B2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{B2}}{\partial u_j}(\mathbf{u}) = p_j \frac{s}{K u_j}, \\
\mathrm{smax}'_{j,C1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{C1}}{\partial u_j}(\mathbf{u}) = p_j \left(1 + \alpha(u_j - s)\right), \\
\mathrm{smax}'_{j,C2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{C2}}{\partial u_j}(\mathbf{u}) = p_j \left(1 + \alpha\left(1 - \frac{s}{u_j}\right)\right),
\end{aligned} \tag{7}$$

where s is the value of smax at $\mathbf{u}$ (always the same smooth max type is used as on the corresponding left hand side).

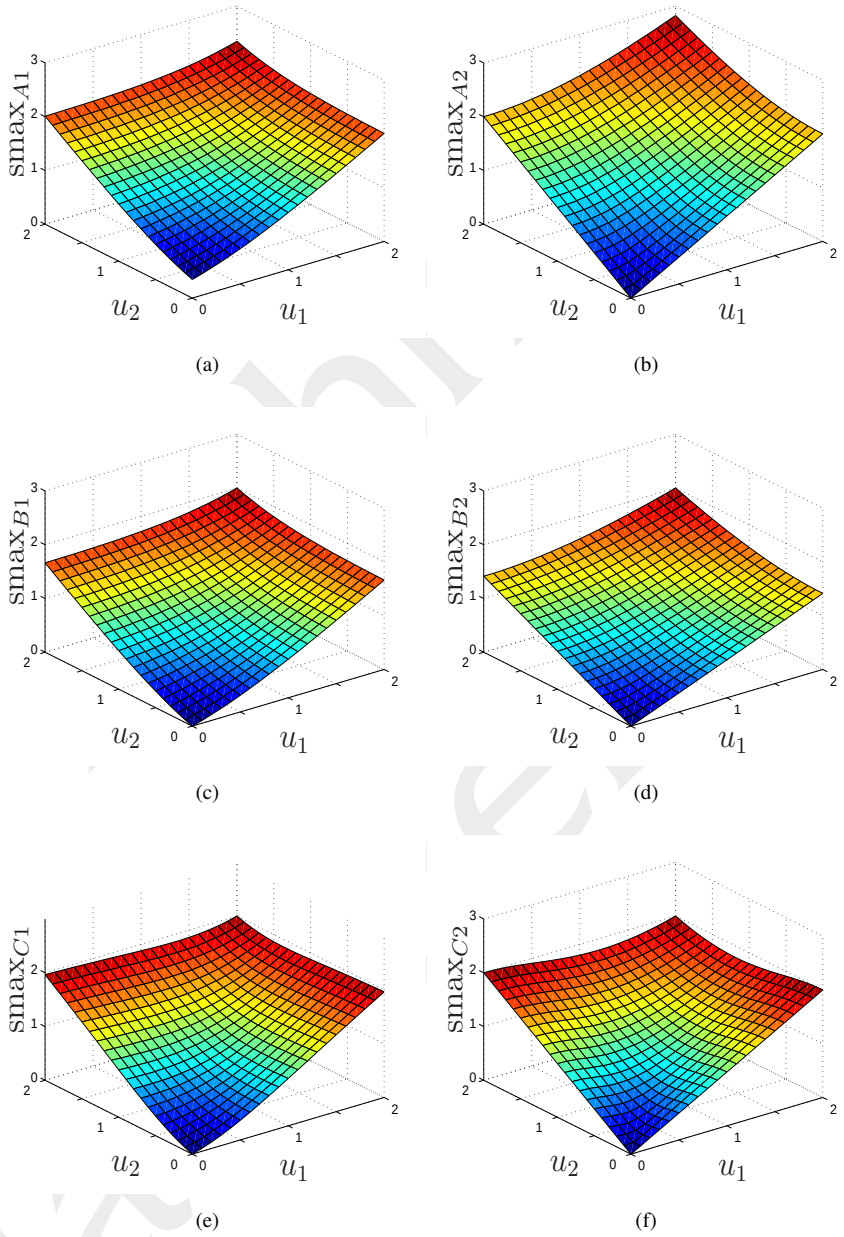


Figure 2: Smooth maximum functions in 2 dimensions ($\alpha = 2$).

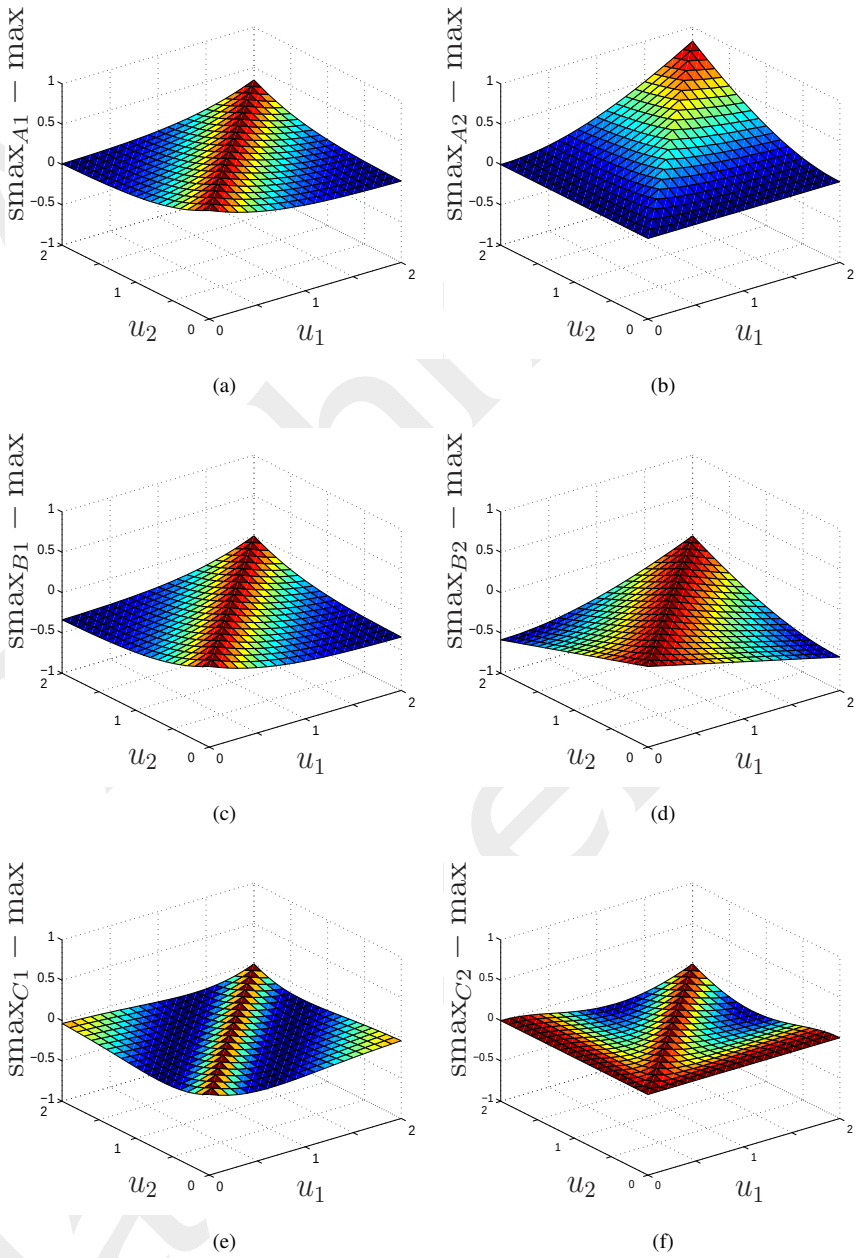


Figure 3: The error of smooth maximum functions in 2 dimensions ($\alpha = 2$).

Interestingly, each derivative contains the factor p_j. In the case of power function based approximations (smax_{A2}, smax_{B2} and smax_{C2}), the derivative also depends on the ratio of the approximated maximum and the j-th number. In the case of smax_{C1}, the derivative also depends on the difference of the approximated maximum and the j-th number.

The second partial derivatives are the following (j,$k = 1,\ldots,K$):

$$
\begin{aligned}
\mathrm{smax}''_{jk,A1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{A1}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\alpha, \\
\mathrm{smax}''_{jk,A2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{A2}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\frac{(\alpha-1)s}{u_j u_k}, \\
\mathrm{smax}''_{jk,B1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{B1}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\alpha, \\
\mathrm{smax}''_{jk,B2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{B2}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\frac{(\alpha-1)s}{K^2 u_j u_k}, \\
\mathrm{smax}''_{jk,C1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{C1}}{\partial u_j \partial u_k}(\mathbf{u}) = \left(-p_j s'_k - p_k s'_j + \delta_{jk}(s'_j + p_j)\right)\alpha, \\
\mathrm{smax}''_{jk,C2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{C2}}{\partial u_j \partial u_k}(\mathbf{u}) = \left(-\frac{p_j s'_k}{u_j} - \frac{p_k s'_j}{u_k} + \delta_{jk}\left(\frac{s'_j}{u_j} + \frac{p_j s}{u_j^2}\right)\right)\alpha,
\end{aligned}
\tag{8}
$$

where $\delta_{jk} = I\{j = k\}$ is the Kronecker delta symbol and s'_j is the value of $\frac{\partial \mathrm{smax}}{\partial u_j}$ at $\mathbf{u}$ (always the same smooth max type is used as on the corresponding left hand side).

4. Smooth maximum based training

A large family of training algorithms can be introduced for convex polyhedron classifiers with the help of smooth maximum functions. One branching point is what smooth maximum type to use. Another is how to approximate the convex polyhedron classifier itself.

Let us introduce the notation $\mathbf{z} = [z_1,\ldots,z_K] = [\mathbf{w}_1^T\mathbf{x} + b_1,\ldots,\mathbf{w}_K^T\mathbf{x} + b_K]$. Three equivalent forms of the convex polyhedron classifier are:

$$
\begin{aligned}
g(\mathbf{x}) &= \mathrm{th}(\min\{z_1,\ldots,z_K\}) \\
&= \min\{\mathrm{th}(z_1),\ldots,\mathrm{th}(z_K)\} \\
&= \min\{\mathrm{th}(z_1) - 1,\ldots,\mathrm{th}(z_K) - 1\} + 1
\end{aligned}
$$

Using the maximum function the previous formulae can be written as

$$\begin{aligned} g(\mathbf{x}) &= \mathrm{th}(-\max\{-z_1,\ldots,-z_K\}) \\ &= -\max\{-\mathrm{th}(z_1),\ldots,-\mathrm{th}(z_K)\} \\ &= -\max\{1-\mathrm{th}(z_1),\ldots,1-\mathrm{th}(z_K)\}+1. \end{aligned}$$

Note that in the third case we always take the maximum of positive numbers.

Now we are ready to introduce smooth versions of g: $\max$ can be replaced with a smooth max and $\mathrm{th}(\gamma)$ with $\mathrm{sgm}(\gamma) = 1/(1+\exp(-\gamma))$ or $\gamma + 0.5$, where sgm is called the logistic sigmoid function. After filtering out some irrelevant combinations we get the following smooth versions of g:

$$\begin{aligned} h_A(\mathbf{x}) &= \mathrm{sgm}(-\mathrm{smax}(-z_1,\ldots,-z_K)), \\ h_B(\mathbf{x}) &= -\mathrm{smax}(-z_1,\ldots,-z_K)+0.5, \\ h_C(\mathbf{x}) &= -\mathrm{smax}(1-\mathrm{sgm}(z_1),\ldots,1-\mathrm{sgm}(z_K))+1. \end{aligned}$$

In the first two cases, smax takes value from $\{\mathrm{smax}_{A1},\mathrm{smax}_{B1},\mathrm{smax}_{C1}\}$. In the third case, smax takes value from $\{\mathrm{smax}_{A1},\mathrm{smax}_{A2},\mathrm{smax}_{B1},\mathrm{smax}_{B2},\mathrm{smax}_{C1},\mathrm{smax}_{C2}\}$.

It will be useful to unify the three branches by decomposing h functions into three parts:

$$h(\mathbf{x}) = h_2(\mathrm{smax}(h_1(\mathbf{z_1}),\ldots,h_1(\mathbf{z_K}))),$$

where h_1 and h_2 are $\mathbb{R} \mapsto \mathbb{R}$ mappings. The h_1 and h_2 parts of the given h functions are the following:

$$\begin{aligned} h_{A1}(z) &= -z, & h_{A2}(s) &= \mathrm{sgm}(-s), \\ h_{B1}(z) &= -z, & h_{B2}(s) &= -s+0.5, \\ h_{C1}(z) &= 1-\mathrm{sgm}(z), & h_{C2}(s) &= -s+1. \end{aligned} \tag{9}$$

The first and the second derivatives of the above functions are:

$$\begin{aligned} h'_{A1}(z) &= -1, & h'_{A2}(s) &= -h_{A2}(s)(1-h_{A2}(s)), \\ h'_{B1}(z) &= -1, & h'_{B2}(s) &= -1, \\ h'_{C1}(z) &= -h_{C1}(z)(1-h_{C1}(z)), & h'_{C2}(s) &= -1, \end{aligned} \tag{10}$$

$$\begin{aligned} h''_{A1}(z) &= 0, & h''_{A2}(s) &= -h'_{A2}(s)(1-2h_{A2}(s)), \\ h''_{B1}(z) &= 0, & h''_{B2}(s) &= 0, \\ h''_{C1}(z) &= h'_{C1}(z)(1-2h_{C1}(z)), & h''_{C2}(s) &= 0. \end{aligned} \tag{11}$$

Let us denote the output of h for input $\mathbf{x}$ by $a = h(\mathbf{x})$. The error of the classifier on example $(\mathbf{x},y)$ can be measured with differentiable loss functions. Two possible choices are the squared loss and the logistic loss:

$$\begin{aligned} \mathrm{loss}_S(a,y) &= \tfrac{1}{2}(a-y)^2, \\ \mathrm{loss}_L(a,y) &= -\ln\left(a^y(1-a)^{1-y}\right). \end{aligned} \tag{12}$$

In the first case, h takes value from $\{h_A,h_B,h_C\}$. In the second case, a has to fall into $[0,1]$, therefore h takes value from $\{h_A,h_C\}$, but if $h = h_C$, then the smooth maximum function cannot be smax_{A1} or smax_{A2}.

The first and the second derivatives of the proposed loss functions with respect to a are:

$$\begin{aligned} \mathrm{loss}'_S(a,y) &= \frac{\partial \mathrm{loss}_S}{\partial a}(a,y) = a-y, \\ \mathrm{loss}'_L(a,y) &= \frac{\partial \mathrm{loss}_L}{\partial a}(a,y) = \frac{1-y}{1-a} - \frac{y}{a}, \end{aligned} \tag{13}$$

$$\begin{aligned} \mathrm{loss}''_S(a,y) &= \frac{\partial^2 \mathrm{loss}_S}{\partial^2 a^2}(a,y) = 1, \\ \mathrm{loss}''_S(a,y) &= \frac{\partial^2 \mathrm{loss}_L}{\partial^2 a^2}(a,y) = \frac{1-y}{(1-a)^2} - \frac{y}{a^2}. \end{aligned} \tag{14}$$

Based on the per example loss, the regularized total loss can be defined as

$$\mathcal{L}(b_1,\mathbf{w}_1,\ldots,b_K,\mathbf{w}_K) = \left(\sum_{i=1}^{n} \mathrm{loss}(h(\mathbf{x}_i),y_i)\right) + \lambda\left(\frac{1}{2}\sum_{j=1}^{K}\mathbf{w}_j^T\mathbf{w}_j\right), \tag{15}$$

where $\mathrm{loss} \in \{\mathrm{loss}_S,\mathrm{loss}_L\}$, and λ is called regularization coefficient. The number of allowed choices for $(\mathrm{smax},h,\mathrm{loss})$ is 19. In every case, a local minimum of $\mathcal{L}$ can be found by derivative based algorithms. This proposed approach of training convex polyhedron classifiers will be referred as SMAX from now.

It is important to note that smooth approximations are used only during the training. In the classification phase, the original formula of the convex polyhedron classifier is applied. Obviously, using different prediction formulae at training and classification may deteriorate the accuracy. A possible way to to handle this problem is to gradually decrease the smoothness of the approximation during the training by increasing the value of α.

The first proposed training method uses stochastic gradient descent for the approximate minimization of $\mathcal{L}$. The pseudo-code of the algorithm can be seen in Figure (4).

Input: $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n)$ // the training set
Input: smax, α, h, loss, K, R, E, B, η, μ, λ, A_0, A_1 // meta-parameters
Output: $(\mathbf{w}_1,b_1),\ldots,(\mathbf{w}_K,b_K)$ // the trained model

$(\mathbf{w}_1,b_1),\ldots,(\mathbf{w}_K,b_K) \leftarrow$ uniform random numbers from $[-R,R]$
// initialization
$(\mathbf{w}_1^{\text{old}},b_1^{\text{old}}),\ldots,(\mathbf{w}_K^{\text{old}},b_K^{\text{old}}) \leftarrow (\mathbf{w}_1,b_1),\ldots,(\mathbf{w}_K,b_K)$
$(\mathbf{w}'_1,b'_1),\ldots,(\mathbf{w}'_K,b'_K) \leftarrow$ zeros

macro AccumlateGradient(i) **begin**
    for $j \leftarrow 1$ **to** K **do** $z_j \leftarrow \mathbf{w}_j^T\mathbf{x}_i + b_j$ // calculate branch activations
    $\mathbf{u} \leftarrow [h_1(z_1),\ldots,h_1(z_K)]^T$
    $s \leftarrow \text{smax}(\mathbf{u})$
    $a \leftarrow h_2(s)$ // calculate answer
    for $j \leftarrow 1$ **to** K **do** // update gradient
        $c'_j \leftarrow \text{loss}'(a,y_i) \cdot h'_2(s) \cdot \text{smax}'_j(\mathbf{u}) \cdot h'_1(z_j)$
        $\mathbf{w}'_j \leftarrow \mathbf{w}'_j + c'_j\mathbf{x}_i + \lambda\mathbf{w}_j/n$
        $b'_j \leftarrow b'_j + c'_j$
    end
end

for $e \leftarrow 1$ **to** E **do** // for all epochs
    $\alpha \leftarrow A_1\alpha + A_0$ // update smoothness
    for $i \leftarrow 1$ **to** n **do** // for all examples
        AccumlateGradient(i)
        if $i \equiv 0 \pmod{B}$ **then** // update model
            for $j \leftarrow 1$ **to** K **do**
                $\Delta \leftarrow \mathbf{w}_j - \mathbf{w}_j^{\text{old}}, \quad \mathbf{w}_j^{\text{old}} \leftarrow \mathbf{w}_j$
                $\mathbf{w}_j \leftarrow \mathbf{w}_j - \eta\mathbf{w}'_j + \mu\Delta$
                $\Delta \leftarrow b_j - b_j^{\text{old}}, \quad b_j^{\text{old}} \leftarrow b_j$
                $b_j \leftarrow b_j - \eta b'_j + \mu\Delta$
            end
            $(\mathbf{w}'_1,b'_1),\ldots,(\mathbf{w}'_K,b'_K) \leftarrow$ zeros // reset gradient
        end
    end
end

Figure 4: Stochastic gradient descent with momentum for training the convex polyhedron classifier.

The meanings of the algorithm's meta-parameters are as follows:

- $\mathrm{smax} \in \{\mathrm{smax}_{A1}, \mathrm{smax}_{A2}, \mathrm{smax}_{B1}, \mathrm{smax}_{B2}, \mathrm{smax}_{C1}, \mathrm{smax}_{C2}\}$: smooth max function,
- $\alpha \in \mathbb{R}$: initial value of the smoothness parameter,
- $h \in \{h_A, h_B, h_C\}$: smooth replacement of g,
- $\mathrm{loss} \in \{\mathrm{loss}_S, \mathrm{loss}_L\}$: per example loss function,
- $K \in \mathbb{N}$: number of hyperplanes in the convex polyhedron classifier,
- $R \in \mathbb{R}$: range of random number generation at model initialization,
- $E \in \mathbb{N}$: number of epochs (iterations over the training set),
- $B \in \mathbb{N}$: batch size — the model is updated after each B example,
- $\eta \in \mathbb{R}$: learning rate — step size at model update,
- $\mu \in \mathbb{R}$: momentum factor — the weight of the previous update in the current one,
- $\lambda \in \mathbb{R}$: regularization coefficient — how aggressively the weights are pushed towards 0,
- $A_0, A_1 \in \mathbb{R}$: coefficients for controlling the change of α.

The time requirement of one iteration is $O(ndK)$, and the time requirement of the algorithm is $O(EndK)$, therefore the algorithm can be run on very large problems. In practice it is not always necessary to find a local minimum. It is often enough to reach a sufficiently small objective function value. Of course, there is no guarantee that the trained model will be acceptable after a modest number of iterations, but at least we are able to test it.

The second proposed training algorithm uses Newton's method for the approximate minimization of $\mathcal{L}$. The pseudo-code of the algorithm can be seen in Figure (5). The meta-parameters of the algorithm are the same as before except that there is no batch size B, learning rate η, and momentum factor μ, and there is a new parameter S, the number of step sizes tried before model update. The role of parameter S is to make the algorithm more stable.

The time requirement of one iteration is $O(nd^2K^2 + d^3K^3)$ and the time requirement of the algorithm is $O(End^2K^2 + Ed^3K^3)$. An advantage of Newton's method over stochastic gradient descent is better accuracy. A disadvantage is the substantially increased time complexity of iterations. It may happen that we are unable to run even one iteration.

It is also true that Newton's method is typically less robust than gradient method. It is more sensible to stuck in minor local minima, and also it is more prone to diverge. A possible way to overcome these difficulties is to introduce a hybrid approach that starts the minimization with gradient method, and then switches to Newton's method.

Input: $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n)$ // the training set
Input: smax, α, h, loss, K, R, E, λ, A_0, A_1, S // meta-parameters
Output: $(\mathbf{w}_1,b_1),\ldots,(\mathbf{w}_K,b_K)$ // the trained model

$(\mathbf{w}_1,b_1),\ldots,(\mathbf{w}_K,b_K) \leftarrow$ uniform random numbers from $[-R,R]$ // initialization
$(\mathbf{w}'_1,b'_1),\ldots,(\mathbf{w}'_K,b'_K)$, $\mathbf{H}$, $\mathbf{g} \leftarrow$ zeros
$\mathcal{L}_{\min} \leftarrow \infty$

macro AccumlateHessian(i) **begin**
  $\mathbf{x}_{i0} \leftarrow 1$ // consider the 0-th coordinate as 1
  for j, k **in** $\{1,\ldots K\} \times \{1,\ldots K\}$ **do**
    $c''_{jk} \leftarrow \mathrm{loss}''(a,y_i) \cdot h'_2(s)^2 \cdot \mathrm{smax}'_j(\mathbf{u})\mathrm{smax}'_k(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
      $\mathrm{loss}'(a,y_i) \cdot h''_2(s) \cdot \mathrm{smax}'_j(\mathbf{u})\mathrm{smax}'_k(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
      $\mathrm{loss}'(a,y_i) \cdot h'_2(s) \cdot \mathrm{smax}''_{jk}(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
      $\mathrm{loss}'(a,y_i) \cdot h'_2(s) \cdot \mathrm{smax}'_j(\mathbf{u})\delta_{jk} \cdot h''_1(z_j)\delta_{jk}$

    for l, m **in** $\{0,\ldots,d\} \times \{0,\ldots,d\}$ **do** // update Hessian
      $\widehat{j} \leftarrow (j-1)(d+1)+l+1$
      $\widehat{k} \leftarrow (k-1)(d+1)+m+1$
      $h_{\widehat{j}\widehat{k}} \leftarrow h_{\widehat{j}\widehat{k}} + c''_{jk}x_{il}x_{im} + \lambda\delta_{l0}\delta_{m0}$
    end
  end
end

for $e \leftarrow 1$ **to** E **do** // for all epochs
  $\alpha \leftarrow A_1\alpha + A_0$ // update smoothness

  for $i \leftarrow 1$ **to** n **do** // for all examples
    AccumlateGradient(i)
    AccumlateHessian(i)
  end

  $\mathbf{v} \leftarrow [(b_1w_{11}\cdots w_{1d})\cdots(b_Kw_{K1}\cdots w_{Kd})]^T$
  $\mathbf{g} \leftarrow [(b'_1w'_{11}\cdots w'_{1d})\cdots(b'_Kw'_{K1}\cdots w'_{Kd})]^T$
  for σ **in** $\{1,2^{-1},\ldots,2^{-S+2},0\}$ **do** // try S step sizes
    $\mathbf{v}_{\mathrm{new}} \leftarrow \mathbf{v} - \sigma\mathbf{H}^{-1}\mathbf{g}$
    $\mathcal{L}_{\mathrm{new}} \leftarrow \mathcal{L}(\mathbf{v}_{\mathrm{new}})$ // use (15)
    if $\mathcal{L}_{\mathrm{new}} < \mathcal{L}_{\min}$ **then** $\mathcal{L}_{\min} \leftarrow \mathcal{L}_{\mathrm{new}}$, $\mathbf{v}_{\mathrm{best}} \leftarrow \mathbf{v}_{\mathrm{new}}$
  end
  $[(b_1w_{11}\cdots w_{1d})\cdots(b_Kw_{K1}\cdots w_{Kd})] \leftarrow \mathbf{v}_{\mathrm{best}}^T$ // update model
  $(\mathbf{w}'_1,b'_1),\ldots,(\mathbf{w}'_K,b'_k)$, $\mathbf{H}$, $\mathbf{g} \leftarrow$ zeros // reset gradient and Hessian
end

Figure 5: Newton's method for training the convex polyhedron classifier.

4.1. Algorithms for regression

Convex polyhedron regression can be introduced analogously with convex polyhedron classification. A *convex K-polyhedron predictor* is a function $g : \mathbb{R}^d \mapsto \mathbb{R}$ that can be written in the following form:

$$\begin{aligned} g(\mathbf{x}) &= \min\{\mathbf{w}_1^T\mathbf{x} + b_1, \ldots, \mathbf{w}_K^T\mathbf{x} + b_K\} \\ &= -\max\{-\mathbf{w}_1^T\mathbf{x} - b_1, \ldots, -\mathbf{w}_K^T\mathbf{x} - b_K\}. \end{aligned} \tag{16}$$

The only difference from convex the K-polyhedron classifier is that the threshold function is missing. The reason behind using the term "convex polyhedron" here is that the set $\{(\mathbf{x},y) \in \mathbb{R}^{d+1} : y \leq g(\mathbf{x})\}$ is a convex polyhedron in $\mathbb{R}^{d+1}$.

Like in the case of classification, it is possible define smooth maximum based algorithms for training. The only difference is that now the only reasonable choice for h and loss is h_B and loss_S, because the target takes value from $\mathbb{R}$. Apart from this restriction, the training algorithms remain the same.

Note that in the case of regression we always have to evaluate all scalar products at prediction. Therefore, unlike the case of classification, there is no extra speedup in the prediction phase, however, the prediction is still not slow. Applying a convex polyhedron predictor can be a reasonable choice, if we know a priori that the optimal predictor g^* is convex.

4.2. Algorithms for collaborative filtering

In the case of collaborative filtering we can obtain a convex polyhedron approach via the generalization of matrix factorization. The answer of the convex polyhedron predictor for user u and item i is

$$g(u,i) = b_u + c_i - \max\left\{-\left(\sum_{l=1}^{L} p_{ul}^{(1)} q_{li}\right), \ldots, -\left(\sum_{l=1}^{L} p_{ul}^{(K)} q_{li}\right)\right\},$$

where $\mathbf{P}^{(k)} \in \mathbb{R}^{N_U \times L}$, $[\mathbf{P}^{(k)}]_{ul} = p_{ul}^{(k)}$, $k = 1, \ldots, K$ called user factor matrices, $\mathbf{Q} \in \mathbb{R}^{L \times N_I}$, $[\mathbf{Q}]_{li} = q_{li}$ called item factor matrix, $\mathbf{b} \in \mathbb{R}^{N_U}$ called user bias vector, and $\mathbf{c} \in \mathbb{R}^{N_I}$ called item bias vector are the parameters of the model.

An analogous variant can be obtained, if we have one user factor matrix $\mathbf{P}$ and K item factor matrices $\mathbf{Q}^{(1)}, \ldots, \mathbf{Q}^{(K)}$:

$$g(u,i) = b_u + c_i - \max\left\{-\left(\sum_{l=1}^{L} p_{ul} q_{li}^{(1)}\right), \ldots, -\left(\sum_{l=1}^{L} p_{ul} q_{li}^{(K)}\right)\right\}.$$

Let us assume the first variant and introduce the notation $\mathbf{z} = [z_1,\ldots,z_K]$, $z_k = \sum_{l=1}^{L} p_{ul}^{(k)} q_{li}$ $(k = 1,\ldots,K)$. The smooth version of g can be obtained as

$$h(u,i) = b_u + c_i + \mathrm{smax}(\mathbf{z}),$$

where $\mathrm{smax} \in \{\mathrm{smax}_{A1},\mathrm{smax}_{B1},\mathrm{smax}_{C1}\}$.

Now it is possible measure the error at example (u,i) with a differentiable loss function:

$$\mathcal{L}_{ui}(\mathbf{w}) = \frac{1}{2}\left(h(u,i) - r_{ui}\right)^2 + \lambda_U \frac{1}{2}\sum_{k=1}^{K}\sum_{l=1}^{L}\left(p_{ul}^{(k)}\right)^2 + \lambda_I \frac{1}{2}\sum_{l=1}^{L}\left(q_{li}\right)^2,$$

where $\mathbf{w}$ denotes the vector containing all parameters of the model ($\mathbf{P}^{(1)},\ldots,\mathbf{P}^{(K)},\mathbf{Q},\mathbf{b},\mathbf{c}$), and λ_U, λ_I are the regularization coefficients. The total loss on the training set is the sum of the per example losses:

$$\mathcal{L}(\mathbf{w}) = \sum_{(u,i)\in\mathcal{T}} \mathcal{L}_{ui}(\mathbf{w}).$$

Similarly to classification and regression, the approximate minimization of $\mathcal{L}$ can be done with stochastic gradient descent. This approach of training the convex polyhedron predictor will be referred as $\mathrm{SMAX}_{\mathrm{CF}}$. Note that in the case of collaborative filtering the typical problem size is large (say $N_U,N_I > 1000$, $L > 10$), therefore Newton's method is computationally too expensive.

The partial derivatives of $\mathcal{L}_{ui}$ can be written as

$$\begin{aligned}
\frac{\partial \mathcal{L}_{ui}}{\partial p_{ul}^{(k)}}(\mathbf{w}) &= \left(h(u,i) - r_{ui}\right)\left(\mathrm{smax}_k'(\mathbf{z}) q_{ui}\right) + \lambda_U p_{ul}^{(k)}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial q_{li}}(\mathbf{w}) &= \left(h(u,i) - r_{ui}\right)\left(\sum_{k=1}^{K} \mathrm{smax}_k'(\mathbf{z}) p_{ul}^{(k)}\right) + \lambda_I q_{li}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial b_u}(\mathbf{w}) &= h(u,i) - r_{ui}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial c_i}(\mathbf{w}) &= h(u,i) - r_{ui},
\end{aligned} \tag{17}$$

Note that the second equation builds upon the assumption $\mathrm{smax} \in \{\mathrm{smax}_{A1},\mathrm{smax}_{B1},\mathrm{smax}_{C1}\}$, and it would *not* be true, if smax was an arbitrary differentiable function.

The pseudo-code of stochastic gradient descent based training can be seen in Figure (6). The meanings of the meta-parameters are the same as before, except that now we have different learning rate and regularization coefficient for users and items. The role of parameter D is to control whether ordering by date within user ratings should be used.

Input: $r_{ui} : (u,i) \in \mathcal{T}, |\mathcal{T}| = n$ // the training set
Input: smax, α, K, R, E, η_U, η_I, λ_U, λ_I, D, A_0, A_1 // meta-parameters
Output: $\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(K)}, \mathbf{Q}$ // the trained model

$\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(K)}$, $\mathbf{Q}$, $\mathbf{b}$, $\mathbf{c} \leftarrow$ uniform random numbers from $[-R,R]$
// initialization

for $e \leftarrow 1$ **to** E **do** // for all epochs
  $\alpha \leftarrow A_1\alpha + A_0$ // update smoothness

  for $u \leftarrow 1$ **to** N_U **do** // for all users
    $\mathcal{T}_u \leftarrow \{i : \exists u : (u,i) \in \mathcal{T}\}$
    $\mathcal{I} \leftarrow$ a random permutation of the elements of $\mathcal{T}_u$
    if $D = 1$ **and** *dates are available for ratings* **then**
      $\mathcal{I} \leftarrow$ the elements of $\mathcal{T}_u$ sorted by rating date (in ascending order)
    end

    for i **in** $\mathcal{I}$ **do** // for user's ratings
      for $k \leftarrow 1$ **to** K **do** $z_k \leftarrow \sum_{l=1}^{L} p_{ul}^{(k)} q_{li}$
      for $k \leftarrow 1$ **to** K **do** $s'_k \leftarrow \mathrm{smax}'_k(-\mathbf{z})$
      $a \leftarrow b_u + c_i - \mathrm{smax}(-\mathbf{z})$ // calculate answer
      $\varepsilon \leftarrow a - y_i$ // calculate error

      $b_u \leftarrow b_u - \eta_U \varepsilon$ // update biases
      $c_i \leftarrow c_i - \eta_I \varepsilon$

      for $l \leftarrow 1$ **to** L **do** // update factors
        $p \leftarrow \sum_{k=1}^{K} s'_k p_{ul}^{k}$
        for $k \leftarrow 1$ **to** K **do** $p_{ul}^{(k)} \leftarrow p_{ul}^{(k)} - \eta_U(\varepsilon s'_k q_{li} + \lambda_U p_{ul}^{(k)})$
        $q_{li} \leftarrow q_{li} - \eta_I(\varepsilon p + \lambda_I q_{li})$
      end
    end
  end
end

Figure 6: Stochastic gradient descent for training the convex polyhedron predictor.

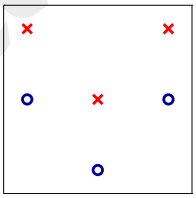


Figure 7: The TOY dataset.

5. Experiments

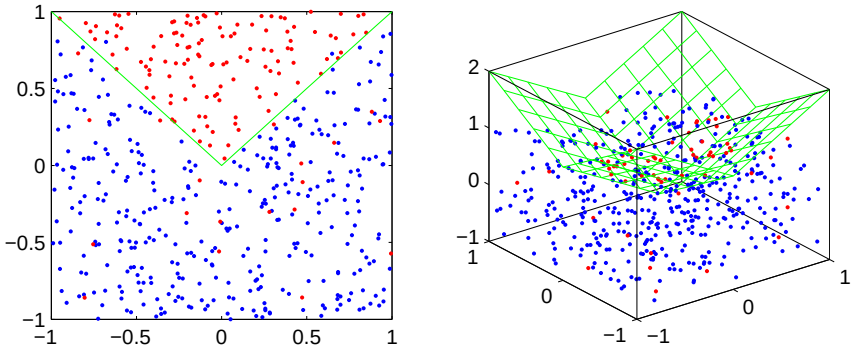
5.1. Classification

In this section, we will compare convex polyhedron classification algorithms with other methods both on artificial and real-life datasets. The artificial datasets involved in the experiments are the following:

- **TOY**: This dataset contains 6 examples: $\mathbf{x}_1 = [-1,2], y_1 = 1$, $\mathbf{x}_2 = [0,1], y_2 = 1$, $\mathbf{x}_3 = [1,2], y_3 = 1$, $\mathbf{x}_4 = [-1,1], y_4 = 0$, $\mathbf{x}_5 = [0,0], y_5 = 0$, $\mathbf{x}_6 = [1,2], y_6 = 0$. The classes can be separated from each other with 2 lines (see Figure 7). One may find it interesting to analyze the differences between the many proposed training algorithms on such an extremely simple dataset.
- **V2**: Let us define the V distribution as the following: The components of the input vector $\mathbf{X}$ are drawn independently, according to uniform distribution over $[-1, +1]$. If $X_d \geq \sum_{j=1}^{d-1} |X_j|$, then the class label Y is set to 1, otherwise it is set to 0. Finally, the value of Y is flipped with probability α. Note, that the Bayes classifier for the V distribution is a convex 2^{d-1}-polyhedron classifier. The V2 dataset contains $n = 10^5$ examples generated according to the V distribution with settings $d = 2$ and $\alpha = 0.05$ (see Figure 8).
- **V3**: The 3-dimensional ($d = 3$, $n = 10^5$) version of the previous dataset (see Figure 8).

The real-life datasets involved in the experiments were extracted from the UCI machine learning repository [2]. Convex polyhedron classification assumes two classes, therefore all problems were transformed to binary ones by merging classes. The specific datasets were the following:

- **ABALONE**: Here the task is to predict from various physical characteristics (e.g. length, diameter, height) whether the number of rings of an abalone is greater than 12. The number of input features in the dataset after variable encoding is $d = 10$, and the number of of examples is $n = 4177$ (16.6 % of the examples belong to the positive



(a) (b)

Figure 8: The V distribution with settings $d = 2$, $\alpha = 0.05$ (a) and $d = 3$, $\alpha = 0.05$ (b). The optimal decision boundary is indicated with green.

class).

- **BLOOD**: This dataset originates from the donor database of Blood Transfusion Service Center in Hsin-Chu City, Taiwan. The goal is to predict whether a donor donated blood at a given date in 2007. The dataset contains $d = 4$ input features (months since last donation, total number of donations, total blood donated in c.c., months since first donation), and $n = 748$ examples (23.8 % positives).
- **CHESS**: This dataset came from the domain of chess endgames. The $d = 6$ input features are integers, representing the location of the white king, the white rook and the black king. The task is to decide whether black can escape from being mated in 14 (or less) moves. The number of examples is $n = 28056$ (9.1 % positives).
- **SEGMENT**: The instances were drawn randomly from a database of 7 outdoor images. The images were hand-segmented to create a classification for every pixel. The task is to predict whether a pixel is part of a window object in the image. The number of features is $d = 19$, and the number of examples is $n = 2310$ (14.3 % positives).

The classification algorithms included in the comparison were the following:

- **FDA**: Regularized Fisher discriminant analysis [12]. Regularization was done by adding the term $\lambda \mathbf{w}^T \mathbf{w}$ to the denominator of the objective function, where $\mathbf{w} \in \mathbb{R}^d$ denotes the weight vector. The sole parameter of the algorithm is the regularization coefficient λ (default value: 10^{-6}).
- **LOGR**: Logistic regression [26] with L2 regularization applied on the weight vector. The minimization of the objective function was done by Newton's method. The

starting point was the all-zero vector. The algorithm has 2 parameters: the regularization coefficient λ (default value: 10^{-6}) and the number of iterations E.

- **SPER**: The smooth variant of Rosenblatt's perceptron [19] with L2 regularization. The activation function was the logistic sigmoid function. The minimization of the objective function was done by Newton's method, started from the all-zero vector. The algorithm has 2 parameters: the regularization coefficient λ (default value: 10^{-6}) and the number of iterations E.
- **ALN**: Adaptive linear neuron [25] with L2 regularization applied on the weight vector. The only parameter of the algorithm is the regularization coefficient λ (default value: 10^{-6}).
- **LSVM**: Linear support vector machine [6]. The algorithm has one parameter: the tradeoff coefficient C.
- **KNN**: K nearest neighbors [13]. The only parameter of the approach is the number of relevant neighbors K.
- **ID3**: ID3 decision tree [18]. All features were treated as continuous ones. The algorithm has 3 parameters: the number of splitting values tried K (default value: 10), the Laplace smoothing term β, and the information gain threshold $\mathcal{G}_{\min}$.
- **MLP**: Multilayer perceptron [24] with L2 regularization applied on the non-bias weights. The objective function was minimized by batch gradient descent with momentum. The parameters of the algorithm are the regularization coefficient λ (default value: 10^{-6}), the number of hidden units K (default value: 5), the range of random initialization R, the number of epochs E, the learning rate η, and the momentum factor μ.
- **SVM**: Support vector machine [6] with Gaussian kernel. The only parameter of the algorithm is the tradeoff coefficient C.
- **MR**: Convex polyhedron classifier with maximal rejection based training (see page 30). The parameters of the algorithm are the number of hyperplanes K, and the tolerance β.
- **SMAX**: The proposed smooth maximum function based approach for convex polyhedron classification (see page 37). The parameters of the algorithm are the training method (G: gradient method, see page 40, N: Newton's method, see page 42, G+N: start with gradient method, and then continue with Newton's method — default: G+N), the smooth max function (smax_{A1}, smax_{A2}, smax_{B1}, smax_{B2}, smax_{C1}, or smax_{C2} — default: smax_{A1}), the smoothness parameter α (default value: 2), the smoothness change coefficients A_1 and A_0 (default values: $A_1 = 1$, $A_0 = 0$) the h function (h_A, h_B, or h_C — default: h_A), the per example loss function (loss_S or loss_L — default: loss_S), the number of hyperplanes K, the range of random initialization R (default value: 1), the number of epochs in the G phase E, the number of epochs in the N phase E_2, the batch size B (default value: n, which means batch mode), the regularization coefficient λ (default value: 10^{-6}), the learning rate η, the momentum factor μ (default value: 0.95),

and the number of step sizes tried before Newton updates S (default value: 10).

For LSVM and SVM the libsvm [7] implementation was used, via the built-in Python interface. All other algorithms were implemented from scratch in Python [20], using the NumPy [1] module. The hardware environment was a notebook PC with Intel Pentium M 2 GHz CPU and 1 Gb memory. If the value of a parameter is not specified, then the default value is used.

5.1.1. Comparing the variants of SMAX

In these experiments I ran the variants of SMAX training on the TOY dataset. Every valid combination of optimization method, loss, h and smax were tested with the 3 different optimization methods (G, N, G+N). The parameters E (number of epochs in the G phase), η (learning rate), E_2 (number of epochs in the N phase), and R (range of random initialization) were set heuristically via "trial and error" for each setting. The other parameters were kept fixed at their default values. The results can be seen in Table 1 and Table 2. The meaning of the last 3 columns are:

- $\|\mathbf{W}\|$: The Frobenius norm of weight matrix part of the solution ($\sqrt{\sum_{k=1}^{K}\sum_{j=1}^{d} w_{kj}^2}$).
- $\mathcal{L}_{01}$: The number of training examples misclassified by the trained model.
- $\mathcal{L}$: The value of the regularized total loss at the solution.

It can be seen, that the G and the G+N methods were always able to build a classifier that does not err on the training set. In contrast, the N method sometimes converged to local minima with relatively high $\mathcal{L}$ value. This is because gradient descent with momentum is less prone to stuck in local minima than Newton's method (however, it needs more iterations to converge).

If we consider the categories defined by the second and third columns, then we can observe the following:

- **SA** ($\mathrm{loss} = \mathrm{loss}_S$, $h = h_A$): In this category the G+N method required a relatively short G phase. The lowest $\mathcal{L}$ value was achieved by $\mathrm{smax} = \mathrm{smax}_{C1}$.
- **SB** ($\mathrm{loss} = \mathrm{loss}_S$, $h = h_B$): This category required 2 magnitudes smaller learning rates than the other ones. (This is because in the case of h_B the changes of the weights are not "dampened" by the sigmoid function.) The $\|\mathbf{W}\|$ values were relatively small. The G+N method required a relatively long G phase. The lowest $\mathcal{L}$ value was achieved by $\mathrm{smax} = \mathrm{smax}_{C2}$.
- **SC** ($\mathrm{loss} = \mathrm{loss}_S$, $h = h_C$): The $\mathcal{L}$ value was typically higher than in SA and SB. The lowest $\mathcal{L}$ value was achieved by $\mathrm{smax} = \mathrm{smax}_{C2}$.
- **LA** ($\mathrm{loss} = \mathrm{loss}_L$, $h = h_A$): In this category the pure N method was more stable than in the other categories: it was always able to achieve zero misclassifications. The

variant	**loss**	**h**	**smax**	**method**	**parameters**	$\|\mathbf{W}\|$	$\mathcal{L}_{01}$	$\mathcal{L}$
#1	S	A	A1	G	$E = 1000, \eta = 0.1$	9.6	0	0.0478
#2	S	A	A1	N	$E_2 = 10, R = 0.5$	4.8	1	0.3493
#3	S	A	A1	G+N	$E = 50, \eta = 0.1, E_2 = 10$	9.6	0	0.0474
#4	S	A	B1	G	$E = 1000, \eta = 0.1$	9.6	0	0.0487
#5	S	A	B1	N	$E_2 = 10$	9.6	0	0.0483
#6	S	A	B1	G+N	$E = 50, \eta = 0.1, E_2 = 10$	9.6	0	0.0483
#7	S	A	C1	G	$E = 1000, \eta = 0.1$	9.3	0	0.0478
#8	S	A	C1	N	$E_2 = 10, R = 0.5$	9.3	0	0.0449
#9	S	A	C1	G+N	$E = 50, \eta = 0.1, E_2 = 10$	9.3	0	0.0476
#10	S	B	A1	G	$E = 1000, \eta = 0.001$	2.4	0	0.0013
#11	S	B	A1	N	$E_2 = 10$	2.4	0	0.0014
#12	S	B	A1	G+N	$E = 200, \eta = 0.001, E_2 = 10$	2.4	0	0.0013
#13	S	B	B1	G	$E = 1000, \eta = 0.001$	2.4	0	0.0015
#14	S	B	B1	N	$E_2 = 10$	2.4	0	0.0015
#15	S	B	B1	G+N	$E = 100, \eta = 0.001, E_2 = 10$	2.4	0	0.0014
#16	S	B	C1	G	$E = 1000, \eta = 0.001$	2.0	0	0.0011
#17	S	B	C1	N	$E_2 = 10$	0.7	1	0.3533
#18	S	B	C1	G+N	$E = 200, \eta = 0.001, E_2 = 10$	2.0	0	0.0011
#19	S	C	A1	G	$E = 1000, \eta = 0.1$	10.2	0	0.2362
#20	S	C	A1	N	$E_2 = 20$	10.2	0	0.2362
#21	S	C	A1	G+N	$E = 200, \eta = 0.1, E_2 = 10$	10.2	0	0.2362
#22	S	C	A2	G	$E = 1000, \eta = 0.1$	9.5	0	0.0488
#23	S	C	A2	N	$E_2 = 20$	9.5	0	0.0488
#24	S	C	A2	G+N	$E = 50, \eta = 0.1, E_2 = 10$	9.5	0	0.0488
#25	S	C	B1	G	$E = 1000, \eta = 0.1$	10.5	0	0.1442
#26	S	C	B1	N	$E_2 = 10, R = 0.5$	10.4	0	0.1440
#27	S	C	B1	G+N	$E = 100, \eta = 0.1, E_2 = 10$	10.5	0	0.1442
#28	S	C	B2	G	$E = 1000, \eta = 0.1$	8.7	0	0.1628
#29	S	C	B2	N	$E_2 = 20, R = 0.5$	9.8	0	0.1576
#30	S	C	B2	G+N	$E = 200, \eta = 0.1, E_2 = 10$	8.8	0	0.1623
#31	S	C	C1	G	$E = 1000, \eta = 0.1$	10.0	0	0.0714
#32	S	C	C1	N	$E_2 = 10$	4.0	1	0.3665
#33	S	C	C1	G+N	$E = 50, \eta = 0.1, E_2 = 10$	10.0	0	0.0714
#34	S	C	C2	G	$E = 1000, \eta = 0.1$	9.3	0	0.0464
#35	S	C	C2	N	$E_2 = 10, R = 0.5$	4.0	1	0.3117
#36	S	C	C2	G+N	$E = 50, \eta = 0.1, E_2 = 10$	9.3	0	0.0464

Table 1: Results of SMAX training on the TOY dataset (with squared loss).

variant	loss	h	smax	method	parameters	$\|\mathbf{W}\|$	$\mathcal{L}_{01}$	$\mathcal{L}$
#37	L	A	A1	G	$E = 1000, \eta = 0.05$	17.5	0	0.1652
#38	L	A	A1	N	$E_2 = 10$	17.5	0	0.1620
#39	L	A	A1	G+N	$E = 200, \eta = 0.05, E_2 = 10$	17.5	0	0.1620
#40	L	A	B1	G	$E = 1000, \eta = 0.05$	17.5	0	0.1662
#41	L	A	B1	N	$E_2 = 10$	17.5	0	0.1592
#42	L	A	B1	G+N	$E = 50, \eta = 0.05, E_2 = 10$	17.5	0	0.1639
#43	L	A	C1	G	$E = 1000, \eta = 0.05$	17.2	0	0.1643
#44	L	A	C1	N	$E_2 = 10$	17.2	0	0.1508
#45	L	A	C1	G+N	$E = 50, \eta = 0.05, E_2 = 10$	17.2	0	0.1638
#46	L	C	B1	G	$E = 1000, \eta = 0.05$	17.1	0	0.8212
#47	L	C	B1	N	$E_2 = 10, R = 0.5$	11.3	0	1.3688
#48	L	C	B1	G+N	$E = 50, \eta = 0.05, E_2 = 10$	17.1	0	0.8210
#49	L	C	B2	G	$E = 1000, \eta = 0.05$	14.8	0	0.8795
#50	L	C	B2	N	$E_2 = 10, R = 0.5$	7.5	2	3.2423
#51	L	C	B2	G+N	$E = 100, \eta = 0.05, E_2 = 10$	14.8	0	0.8783
#52	L	C	C1	G	$E = 1000, \eta = 0.05$	17.1	0	0.4095
#53	L	C	C1	N	$E_2 = 10, R = 0.5$	8.4	1	1.9603
#54	L	C	C1	G+N	$E = 50, \eta = 0.05, E_2 = 10$	17.1	0	0.4094
#55	L	C	C2	G	$E = 1000, \eta = 0.05$	17.2	0	0.1583
#56	L	C	C2	N	$E_2 = 10, R = 0.5$	8.4	1	1.9607
#57	L	C	C2	G+N	$E = 50, \eta = 0.05, E_2 = 10$	17.3	0	0.1582

Table 2: Results of SMAX training on the TOY dataset (with logistic loss).

lowest $\mathcal{L}$ value was achieved by $\mathrm{smax} = \mathrm{smax}_{C1}$.

- **LC** ($\mathrm{loss} = \mathrm{loss}_L$, $h = h_C$): In this category the pure N method performed poorly in terms of misclassifications. The $\mathcal{L}$ value was typically higher than in LA. The lowest $\mathcal{L}$ value was achieved by $\mathrm{smax} = \mathrm{smax}_{C2}$.

5.1.2. Comparing SMAX with other methods

In the next experiments we will compare the proposed SMAX approach with other classification algorithms. The algorithms were tested on the given problems using 10-fold cross validation. This means that each dataset was partitioned randomly into 10 parts. Then, each algorithm was run on each problem 10 times, so that in the i-th run the i-th part was used as the test set, and the other parts as the training set.

The performance measures used in the experiments were the following:

- **AUC**: The area under the receiver operating characteristics [9]. Given a predictor g and a test dataset $(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_m,y_m)$ the AUC value can be obtained as follows: At first, the values $g(x_1),\ldots,g(x_m)$ are calculated, and sorted in descending order. If we denote the indices of the sorted sequence by $(1),\ldots,(m)$, and introduce the notations

$\mathrm{TP}_0 = 0, \mathrm{FP}_0 = 0, \mathrm{TP}_k = \sum_{i=1}^{k} y_{(i)} / \sum_{i=1}^{m} y_i$, and $\mathrm{FP}_k = \sum_{i=k+1}^{m} y_{(i)} / \sum_{i=1}^{m} (1 - y_i)$ $(k = 1 \ldots, m)$, then the AUC value can be written as:

$$\mathrm{AUC} = \sum_{k=1}^{m} (\mathrm{FP}_k - \mathrm{FP}_{k-1})(\mathrm{TP}_k + \mathrm{TP}_{k-1})/2.$$

In each experiment, we measure the empirical mean and standard deviation of the AUC values obtained from the 10 runs.

- **ΔAUC**: The difference of the AUC value from the value from the AUC of FDA, which is treated as the baseline result. Again, we measure the empirical mean and the standard deviation of the ΔAUC values obtained from the 10 runs. Note that the standard deviation of AUC and ΔAUC provide different information about the uncertainty of the measurement. The second value is typically smaller than the first, and it is more useful for comparing algorithms.
- **TTIME**: Training time in seconds, summed over the 10 runs.
- **VTIME**: Validation time in seconds, summed over the 10 runs.

The results of classification algorithms on the V2 dataset can be seen in Table 3. Not surprisingly, nonlinear methods outperformed linear ones on this problem in terms of AUC. According to the (mean) AUC value, SMAX was the third best algorithm. According to $\Delta\mathrm{AUC}/\mathrm{VTIME}$, it was the best one.

Results on the V3 dataset can be seen in Table 4. The accuracy of the methods is generally lower than in the case of V2 in terms of AUC. This is because now the optimal decision surface is more complex, and the input space is less densely filled with training examples as in previous case. Again, according to the AUC value, SMAX was the third best algorithm, and according to $\Delta\mathrm{AUC}/\mathrm{VTIME}$, it was the best one.

Results for the ABALONE dataset can be seen in Table 5. Although the highest AUC values were achieved by nonlinear methods, the accuracy of linear methods was relatively good. Nevertheless, SMAX was still the best algorithm in terms of $\Delta\mathrm{AUC}/\mathrm{VTIME}$, slightly outperforming SPER. According to the AUC value, SMAX was the third best method. The other convex polyhedron method, MR performed weak on this problem.

Results for the BLOOD dataset can be seen in Table 6. The best accuracies achieved by linear and nonlinear methods were close to each other. Some nonlinear methods (including MR) performed weak. According to the AUC value, SMAX was the third best algorithm. According to $\Delta\mathrm{AUC}/\mathrm{VTIME}$, it was the second best one (beaten by MLP, tied with LOGR).

Results for the CHESS dataset can be seen in Table 7. We can observe that ID3 and KNN show outstanding accuracy. This interesting phenomenon can be explained by the characteristics of the chess endgames domain. Recall that the inputs are 6 integers,

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.877 (±0.017)	+0.000 (±0.000)	0.08	0.006
LOGR	$E = 1$	0.877 (±0.017)	+0.000 (±0.000)	0.45	0.006
SPER	$E = 1$	0.877 (±0.017)	+0.000 (±0.000)	0.47	0.006
ALN		0.877 (±0.017)	+0.000 (±0.000)	0.43	0.006
LSVM	$C = 10$	0.877 (±0.017)	+0.000 (±0.000)	79.0	0.006
KNN	$K = 35$	0.930 (±0.015)	+0.053 (±0.010)	0.00	27.6
ID3	$\beta = 1, \mathcal{G}_{\min} = 0.001$	0.929 (±0.014)	+0.052 (±0.009)	31.6	0.18
MLP	$R = 0.2, E = 200,$ $\eta = 0.0005, \mu = 0.9$	0.923 (±0.014)	+0.046 (±0.005)	120	0.07
SVM	$C = 10$	0.927 (±0.013)	+0.050 (±0.010)	61.5	3.28
MR	$K = 6, \beta = 0.2$	0.920 (±0.014)	+0.043 (±0.006)	0.31	0.010
SMAX	$K = 2, R = 0.1,$ $E = 500, E_2 = 0,$ $\eta = 0.005$	0.925 (±0.013)	+0.048 (±0.007)	54.6	0.009

Table 3: Results of classification algorithms on the V2 dataset.

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.846 (±0.017)	+0.000 (±0.000)	0.08	0.006
LOGR	$E = 1$	0.846 (±0.017)	+0.000 (±0.000)	0.45	0.006
SPER	$E = 1$	0.846 (±0.017)	+0.000 (±0.000)	0.47	0.006
ALN		0.846 (±0.017)	+0.000 (±0.000)	0.43	0.006
LSVM	$C = 10$	0.846 (±0.018)	+0.000 (±0.000)	60.7	0.006
KNN	$K = 35$	0.887 (±0.020)	+0.041 (±0.010)	0.00	27.5
ID3	$\beta = 1, \mathcal{G}_{\min} = 0.001$	0.877 (±0.019)	+0.031 (±0.009)	22.1	0.19
MLP	$R = 0.2, E = 200$ $\eta = 0.0005, \mu = 0.9$	0.877 (±0.016)	+0.031 (±0.005)	120	0.07
SVM	$C = 10$	0.888 (±0.015)	+0.041 (±0.007)	125	3.17
MR	$K = 12, \beta = 0.2$	0.867 (±0.017)	+0.021 (±0.006)	0.47	0.012
SMAX	$K = 6, R = 0.1,$ $E = 500, E_2 = 0,$ $\eta = 0.005$	0.884 (±0.017)	+0.038 (±0.007)	105	0.011

Table 4: Results of classification algorithms on the V3 dataset.

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.844 (±0.018)	+0.000 (±0.000)	0.067	0.004
LOGR	$E = 1$	0.849 (±0.018)	+0.006 (±0.002)	0.206	0.004
SPER	$E = 2$	0.852 (±0.018)	+0.009 (±0.005)	0.256	0.004
ALN		0.849 (±0.018)	+0.006 (±0.002)	0.195	0.004
LSVM	$C = 10$	0.850 (±0.021)	+0.007 (±0.005)	16.47	0.004
KNN	$K = 25$	0.847 (±0.017)	+0.003 (±0.007)	0.000	7.037
ID3	$\beta = 2, \mathcal{G}_{\min} = 0.001$	0.821 (±0.024)	−0.023 (±0.011)	164.2	0.153
MLP	$R = 0.2, E = 500$ $\eta = 0.002, \mu = 0.95$	0.866 (±0.017)	+0.022 (±0.006)	138.1	0.031
SVM	$C = 1000$	0.863 (±0.015)	+0.019 (±0.007)	30.23	1.517
MR	$K = 6, \beta = 0.2$	0.748 (±0.030)	−0.095 (±0.020)	0.481	0.007
SMAX	$K = 5, R = 0.2,$ $E = 5000, E_2 = 5,$ $\eta = 0.01$	0.855 (±0.019)	+0.012 (±0.008)	430.1	0.007

Table 5: Results of classification algorithms on the ABALONE dataset.

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.754 (±0.043)	+0.000 (±0.000)	0.009	0.002
LOGR	$E = 2$	0.755 (±0.044)	+0.001 (±0.007)	0.039	0.002
SPER	$E = 4$	0.754 (±0.043)	+0.000 (±0.008)	0.055	0.002
ALN		0.753 (±0.041)	−0.002 (±0.010)	0.034	0.002
LSVM	$C = 0.1$	0.745 (±0.048)	−0.009 (±0.020)	1.738	0.002
KNN	$K = 35$	0.759 (±0.053)	+0.004 (±0.047)	0.000	0.157
ID3	$\beta = 5, \mathcal{G}_{\min} = 0.005$	0.732 (±0.056)	−0.022 (±0.038)	1.538	0.010
MLP	$R = 0.5, E = 2000$ $\eta = 0.01, \mu = 0.95$	0.768 (±0.053)	+0.013 (±0.024)	93.35	0.008
SVM	$C = 2000$	0.744 (±0.053)	−0.010 (±0.035)	17.74	0.067
MR	$K = 4, \beta = 0.1$	0.711 (±0.067)	−0.043 (±0.052)	0.054	0.003
SMAX	$K = 6, R = 0.2,$ $E = 500, E_2 = 5,$ $\eta = 0.0001$	0.757 (±0.040)	+0.002 (±0.009)	55.64	0.004

Table 6: Results of classification algorithms on the BLOOD dataset.

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.853 (±0.005)	+0.000 (±0.000)	0.353	0.013
LOGR	$E = 5$	0.854 (±0.005)	+0.001 (±0.002)	1.949	0.013
SPER	$E = 6$	0.854 (±0.005)	+0.001 (±0.001)	2.711	0.013
ALN		0.832 (±0.006)	−0.020 (±0.004)	1.282	0.013
LSVM	$C = 0.001$	0.651 (±0.123)	−0.201 (±0.120)	106.9	0.013
KNN	$K = 15$	0.982 (±0.002)	+0.129 (±0.005)	0.000	279.5
ID3	$\beta = 0.1$, $\mathcal{G}_{\min} = 0.001$	0.993 (±0.003)	+0.140 (±0.006)	192.9	0.700
MLP	$R = 0.01$, $E = 500$ $\eta = 0.0002$, $\mu = 0$	0.836 (±0.006)	−0.017 (±0.004)	916.1	0.178
SVM	$C = 100$	0.955 (±0.006)	+0.102 (±0.007)	277.5	18.76
MR	$K = 6$, $\beta = 0.2$	0.916 (±0.008)	+0.063 (±0.008)	0.783	0.026
SMAX	$K = 6$, $R = 0.2$, $E = 1000$, $E_2 = 5$, $\eta = 0.0002$	0.937 (±0.006)	+0.085 (±0.009)	2231	0.026

Table 7: Results of classification algorithms on the CHESS dataset.

representing the coordinates of the pieces, and the task is to decide if black can avoid being mated in 14 moves. All of the given methods except ID3 and KNN base their model upon the linear combination(s) of the features. In chess, this information is not very useful, because the position value is a highly nonlinear function of the coordinates of the pieces (e.g. the relation is not monotonic). If we analyze the performance of SMAX, then we can see that it was the fourth best method in terms of AUC, and it was the best method in terms of $\Delta\mathrm{AUC}/\mathrm{VTIME}$.

Results for the SEGMENT dataset are shown in Table 8. We can see that linear methods were strongly outperformed by nonlinear ones in terms of accuracy on this problem. The only nonlinear method that performed poorly was MLP. According to the AUC value, SMAX was the best algorithm, tied with SVM. According to $\Delta\mathrm{AUC}/\mathrm{VTIME}$, it was the sole best.

Summarizing the results of the experiments, we can say that SMAX proved to be a useful classification algorithm. Typically, it was less accurate than sophisticated nonlinear methods but more accurate than linear methods. Compared to MR, the other convex polyhedron algorithm, SMAX was more accurate in all of the 6 test problems. If take both accuracy and classification speed into account, then SMAX performed particularly well.

A disadvantage of SMAX on the given problems was relatively long training time (however

Method	Parameters	AUC	ΔAUC	TTIME	VTIME
FDA		0.930 (±0.019)	+0.000 (±0.000)	0.077	0.003
LOGR	$E = 10$	0.945 (±0.017)	+0.015 (±0.009)	0.420	0.003
SPER	$E = 10$	0.942 (±0.020)	+0.012 (±0.011)	0.448	0.003
ALN		0.931 (±0.024)	+0.001 (±0.011)	0.127	0.003
LSVM	$C = 10$	0.939 (±0.024)	+0.009 (±0.014)	5.519	0.003
KNN	$K = 15$	0.988 (±0.009)	+0.058 (±0.019)	0.000	2.748
ID3	$\beta = 0.01, \mathcal{G}_{\min} = 0.001$	0.987 (±0.005)	+0.057 (±0.018)	39.09	0.045
MLP	$R = 0.01, E = 500$ $\eta = 5 \cdot 10^{-6}, \mu = 0.95$	0.858 (±0.026)	−0.072 (±0.025)	76.75	0.018
SVM	$C = 5 \cdot 10^5$	0.989 (±0.010)	+0.058 (±0.019)	84.82	0.260
MR	$K = 8, \beta = 0.2$	0.973 (±0.013)	+0.042 (±0.017)	0.342	0.006
SMAX	$K = 6, R = 0.05,$ $E = 500, E_2 = 5,$ $\eta = 0.008$	0.989 (±0.010)	+0.059 (±0.016)	195.6	0.006

Table 8: Results of classification algorithms on the SEGMENT dataset.

it was still acceptable). I emphasize that the complexity of gradient method based SMAX training is $O(EndK)$, therefore the approach is able to deal with very large problems (as it will be demonstrated in the collaborative filtering experiments).

5.1.3. Notes on running times

Because the implementation environment was Python + NumPy, the measured running times not always reflect the true time requirements of the algorithms. The reason why such phenomena can occur is that Python is a relatively slow, interpreted language, while NumPy is a highly optimized library of numerical routines.

In most cases (FDA, LOGR, SPER, ALN, KNN, MLP, MR, SMAX with gradient training), it was possible to translate every important step of the algorithm to linear algebra operations supported by NumPy, and therefore the overhead of using an interpreted language was small.

In other cases (ID3, SMAX with Newton training), there were critical parts written in pure Python, which resulted a significantly increased running time. These algorithms could be speeded up greatly (up to a constant factor only of course), if we implemented them in C/C++.

In the case of the support vector machines (LSVM, SVM), Python was used only as a wrapper. Most of the computation was done by the highly optimized libsvm library, therefore, the measured running times can be considered as "state of the art".

5.2. Collaborative filtering

5.2.1. The NETFLIX dataset

This collaborative filtering dataset is currrently one of the largest publicly available machine learning datasets. It contains about 100 million rating from over 480 thousand users on nearly 18 thousand items (movies). The dataset was provided generously by Netflix, the popular movie rental service, for the Netflix Prize (NP) competition [5].

The examples are (u,i,r,d) quadruplets, representing that user u rated item i as r on date d. The ratings values are integers from 1 to 5, where 1 is the worst, and 5 is the best. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received by Netflix during this period.

The collected data was released in a train–test setting in the following manner: Netflix selected a random subset of users from their entire customer base with at least 20 ratings in the given period. A Hold-out set was created from the 9 most recent ratings of the users, consisting of about 4.2 million ratings. The remaining data formed the Training set. The ratings of the Hold-out set were split randomly with equal probability into three subsets of equal size: Quiz, Test and Probe. The Probe set was added to the Training set and was released with ratings. The ratings of the Quiz and Test sets were withheld as a Qualifying set to evaluate competitors. The Quiz/Test split of the qualifying set is unknown to the public. I remark that the date based partition of the entire NP dataset into train–test sets reflects the original aim of recommender systems, which is the prediction of future interest of users from their past ratings/activities.

As the aim of the competition is to improve the prediction accuracy of user ratings, Netflix adopted RMSE (root mean squared error) as evaluation measure. The goal of the competition is to reduce the RMSE on the Test set by at least 10 percent, relative to the RMSE achieved by Netflix's own system Cinematch.[5] The contestants have to submit predictions for the Qualifying set. The organizers return the RMSE of the submissions on the Quiz set, which is also reported on a public leaderboard.[6] Note that the RMSE on the Test set is withheld by Netflix.

[5]The first team achieving the 10 percent improvement is promised to be awarded by a Grand Prize of $1 million by Netflix. Not surprisingly, this prospective award drawn much interest towards the competition. So far, more than 3 000 teams submitted entries for the competition.

[6]`http://www.netflixprize.com/leaderboard`

There are some interesting characteristics of the data and the set-up of the competition that pose a difficult challenge for prediction:

- The distribution over the time of the ratings of the Hold-out set is quite different from the Training set. As a consequence of the selection method, the Hold-out set does not reflect the skewness of the movie-per-user, observed in the much larger Training set. Therefore the Qualifying set contains approximately equal number of queries for often and rarely rating users.
- The designated aim of the release of the Probe set is to facilitate unbiased estimation of RMSE for the Quiz/Test sets despite of the different distributions of the Training and the Hold-out sets. In addition, it permits off-line comparison of predictors before submission.
- We already mentioned that users' activity at rating is skewed. To put this into numbers, ten percent of users rated 16 or fewer movies and one quarter rated 36 or fewer. The median is 93. Some very active users rated more than 10,000 movies. A similar biased property can be observed for movies: The most-rated movie, *Miss Congeniality* was rated by almost every second user, but a quarter of titles were rated fewer than 190 times, and a handful were rated fewer than 10 times [3].
- The variance of movie ratings is also very different. Some movies are rated approximately equally by the user base (typically well), and some partition the users. The latter ones may be more informative in predicting the taste of individual users.

5.2.2. Comparing $\mathrm{SMAX}_{\mathrm{CF}}$ with other methods

The algorithms involved in the experiments were the following:

- **DC**: Double centering [4]. The only parameter of the algorithm is the number of epochs E (default value: 2).
- **BRISMF**: Biased regularized incremental simultaneous matrix factorization [21]. The parameters of the algorithm are the number of epochs E, the number of factors L, the user learning rate η_U (default value: 0.016), the item learning rate η_I (default value: 0.005), the user regularization coefficient λ_U (default value: 0.015), and the item regularization coefficient λ_I (default value: 0.015).
- **NSVD1**: Item neighbor based approach with factorized similarity (also known as Paterek's NSVD1) [16]s. The parameters of the algorithm are the number of epochs E, the number of factors L, the user learning rate η_U (default value: 0.005), the item learning rate η_I (default value: 0.005), the user regularization coefficient λ_U (default value: 0.015), and the item regularization coefficient λ_I (default value: 0.015).
- $\mathbf{SMAX}_{\mathrm{CF}}$: The proposed smooth maximum based convex polyhedron approach (see page 43). The parameters of the algorithm are the smooth max function (default value: smax_{A1}), the smoothness parameter α (default value: 2), the smoothness change

parameters A_1 and A_0 (default value: $A_1 = 1$, $A_0 = 0.25$), the number of epochs E, the number of factors L, the user learning rate η_U (default value: 0.016), the item learning rate η_I (default value: 0.005), the user regularization coefficient λ_U (default value: 0.015), and the item regularization coefficient λ_I (default value: 0.015).

All algorithms were implemented in C++ from scratch. The hardware environment was a server PC with Intel Pentium Q9300 2.5 GHz CPU and 3 Gb memory.

Let us denote the NETFLIX Training set by $\mathcal{T} = \{(u_1,i_1,r_1,d_1),\ldots,(u_n,i_n,r_n,d_n)\}$, and the Probe set by $\mathcal{P} = \{(u_1,i_1,r_1,d_1),\ldots,(u_m,i_m,r_m,d_m)\}$. The exact sizes of the sets are $n = 100{,}480{,}507$ and $m = 1{,}408{,}395$. All algorithms were trained using $\mathcal{T} \setminus \mathcal{P}$, and then the Probe RMSE of the trained predictor g was calculated as

$$\text{Probe RMSE} = \sqrt{\frac{1}{|\mathcal{P}|}\sum_{(u,i,r,d)\in\mathcal{P}} (g(u,i) - r)^2}.$$

The results of individual algorithms are shown in Table 9. Recall that SMAX_{CF} can be considered as a generalization of BRISMF. We can see, that the SMAX_{CF} approach was able to boost the accuracy of BRISMF, however if we used more factors, then the benefit was smaller. The NSVD1 approach was less accurate than than BRISMF and SMAX_{CF}, and not surprisingly, DC was the worst in terms of RMSE. It is true for all of BRISMF, NSVD1, and SMAX_{CF} that the accuracy was increasing with introducing more factors.

Each experiment consists of three main phases: data loading, training, and validation. The last column of the table shows the total running time of the experiments in seconds. If we take into account that more than 99 million examples were used for training, then we can conclude that all of the presented algorithms are efficient in terms of time requirement.

In the last experiments the predictions of the previous methods for the Probe set were blended with L2 regularized linear regression. The value of the regularization coefficient was $\lambda = 1.4$. The results can be seen in Table 10.

The last column shows the 10-fold cross validation Probe RMSE of the optimal linear combination of the inputs. The reason why the single-input blends (#11, #12, and #13) have lower RMSE than the inputs themselves is that the linear blender introduces a bias term too. We can see that the SMAX_{CF} approach was able to improve the result of the combination of BRISMF and NSVD1 models. This indicates that SMAX_{CF} was able to capture new aspects of the data that was not captured by BRISMF and NSVD1.

No.	Method	Parameters	Probe RMSE	Running time (seconds)
#1	DC		0.9868	11
#2	BRISMF	$L = 10$, $E = 13$	0.9190	161
#3	BRISMF	$L = 20$, $E = 12$	0.9125	263
#4	BRISMF	$L = 50$, $E = 12$	0.9081	598
#5	NSVD1	$L = 10$, $E = 26$	0.9492	568
#6	NSVD1	$L = 20$, $E = 24$	0.9459	1057
#7	NSVD1	$L = 50$, $E = 22$	0.9435	1900
#8	SMAX_{CF}	$L = 10$, $E = 18$	0.9169	861
#9	SMAX_{CF}	$L = 20$, $E = 18$	0.9114	1234
#10	SMAX_{CF}	$L = 50$, $E = 18$	0.9079	2692

Table 9: Results of collaborative filtering algorithms on the NETFLIX dataset.

No.	Inputs	Probe RMSE
#11	#4	0.9069
#12	#7	0.9430
#13	#10	0.9069
#14	#2+#3+#4	0.9065
#15	#5+#6+#7	0.9429
#16	#8+#9+#10	0.9068
#17	#14+#15	0.9035
#18	#14+#16	0.9050
#19	#15+#16	0.9033
#20	#14+#15+#16	0.9021

Table 10: Results of linear blending on the NETFLIX dataset.

6. Conclusion

Convex polyhedron classifiers are special binary classifiers that fit well to unbalanced problems, because they tend to classify negative examples quickly. Despite this appealing property, the approach is not frequently used in practice. The main reason for that is the lack of good training algorithms.

In this paper I proposed novel and computationally efficient algorithms for training convex polyhedron classifiers. The proposed algorithms are based on the smooth approximation of the maximum function. I also introduced the analogous variant of the smooth maximum approach for regression and collaborative filtering.

The usefulness of the proposed methods was demonstrated via experiments on artificial and real datasets. It turned out turned out that smooth maximum based classifiers are able to provide a good tradeoff between accuracy and classification time on unbalanced classification problems. In the case of collaborative filtering, smooth maximum based methods are able to complement other methods well.

References

[1] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant. Numerical Python, 2001. URL: http://www.numpy.org/.

[2] A. Asuncion and D. J. Newman. UCI Machine Learning Repository, 2007. URL: http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[3] R. Bell, Y. Koren, and C. Volinsky. Chasing $1,000,000: How we won the Netflix Progress Prize. *ASA Statistical and Computing Graphics Newsletter*, 18(2):4–12, 2007.

[4] R. M. Bell and Y. Koren. Improved neighborhood-based collaborative filtering. In *Proc. of the KDD Cup and Workshop 2007*, pages 7–14, 2007.

[5] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. of the KDD Cup and Workshop 2007*, pages 3–6, 2007.

[6] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[7] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*. National Taiwan University, 2001. URL: http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[8] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[9] J. P. Egan. *Signal detection theory and ROC analysis*. Academic Press, New York, 1975.

[10] M. Elad, Y. Hel-Or, and R. Keshet. Pattern detection using a maximal rejection classifier. In *Proc. of the 4th International Workshop on Visual Form*, pages 514–524, 2001.

[11] P. Fischer. More or less efficient agnostic learning of convex polygons. In *Proc. of the 8th Annual Conference on Computational Learning Theory*, pages 228–236. ACM Press, New York, 1995.

[12] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[13] E. Fix and J. L. Hodges. Discriminatory analysis: Non-parametric discrimination: Consistency properties. Technical Report 4, US Air Force School of Aviation Medicine, 1951.

[14] A. R. Klivans, R. O'Donnell, and R. A. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):804–840, 2004.

[15] S. Kwek and L. Pitt. PAC learning intersections of halfspaces with membership quieries. *Algorithmica*, 22:53–75, 1998.

[16] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. of the KDD Cup and Workshop 2007*, pages 39–42, 2007.

[17] I. Pilászy and T. Dobrowiecki. Constructing large margin polytope classifiers with a multiclass classification algorithm. In *Proc. of the 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'2007)*, pages 261–264, 2007.

[18] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[19] F. Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, Washington D.C., 1962.

[20] G. van Rossum. An introduction to Python, 2006.
URL: http://www.network-theory.co.uk/docs/pytut/.

[21] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research (Special topic on Mining and Learning with Graphs and Relations)*, 10:623–656, 2009.

[22] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

[23] S. Vempala. A random sampling based algorithm for learning the intersection of half-spaces. In *Proc. of the 38th Annual Symposium on Foundations of Computer Science*, pages 508–513, 1997.

[24] P. J. Werbos. *Beyond regression: New tools for prediction and analysis in the behaviour sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[25] B. Widrow. An adaptive "adaline" neuron using chemical "memistors". Technical Report 1553-2, Stanford Electronics Laboratories, 1960.

[26] E. B. Wilson and J. Worcester. The determination of L.D.50 and its sampling error in bio-assay. *Proceedings of the National Academy of Sciences*, 29:79–85, 1943.