Convex polyhedron learning and its applications

PhD thesis

Gábor Takács

submitted to

Budapest University of Technology and Economics, Budapest, Hungary
Faculty of Electrical Engineering and Informatics

October, 2009

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

gtakacs@mit.bme.hu, gtakacs@sze.hu

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Béla Pataki, my advisor, for initiating me into research and guiding me over the years. I am grateful to him not only for his constant help and valuable advice, but also for his friendly character and good humor.

I would like to thank Gábor Horváth, my teacher, for giving me interesting tasks that greatly influenced my interest. His neural networks course remains an unforgettable part of my undergraduate studies.

I would like to thank István Pilászy, Bottyán Németh, and Domonkos Tikk, the members of team Gravity, for their sincere enthusiasm to machine learning. I still enjoy the discussions with them about scientific and other topics.

I would like to thank Zoltán Horváth, my senior colleague for listening me many times and asking good questions. I am also grateful to him for teaching me some cool mathematical tricks and helping me to meet people with particularly great knowledge.

I would like to thank my parents for bringing me up and encouraging me in my studies. This work would not have been possible without their support.

Finally, I would like to thank my wife, Katalin for her never ending love and patience. She kept me motivated constantly, and comforted me when I was down. I dedicate this thesis to her and to the fruit of our love, the little Ágnes.

Abstract

From a possible engineer's point of view *learning* can be considered as discovering the relationship between the features of a phenomenon. *Machine learning* (data mining) is a variant of learning, in which the observations about the phenomenon are available as data, and the connection between the features is discovered by a program.

In the case of *classification* the phenomenon is modeled by a random pair $(\mathbf{X}, Y)$, where the d-dimensional continuous $\mathbf{X}$ is called input, and the discrete (often binary) Y is called label. In the case of *collaborative filtering* the phenomenon is modeled by a random triplet (U, I, R), where the discrete U is called user identifier, the discrete I is called item identifier, and the continuous R is called rating value.

Unbalanced problems i.e. in which one class label occurs much more frequently than the other form an interesting subset among binary classification problems. In practice such problems often arise for example in the field of medical and technical diagnostics.

A *convex polyhedron classifier* is a function $g : \mathbb{R}^d \mapsto \{+1, -1\}$ with the property that the decision region $\{\mathbf{x} \in \mathbb{R}^d : g(\mathbf{x}) = 1\}$ is a convex polyhedron. At classifying an input $\mathbf{x} \in \mathbb{R}^d$, we have to substitute $\mathbf{x}$ into the linear functions defining the polyhedron. If any of the substitutions gives a negative number, then we can stop the computation immediately, since the class label will be necessarily -1 in this case. As a consequence, convex polyhedron classifiers fit well to unbalanced problems.

The convex polyhedron based approach has its analogous variant for collaborative filtering too. In this case the utility of the approach is that it gives a unique solution of the problem that can be a useful component of a blended solution involving many different models.

A related problem to classification is determining the *convex separability* of point sets. Let us assume that $\mathcal{P}$ and $\mathcal{Q}$ are finite sets in $\mathbb{R}^d$. The task is to decide whether there exist a convex polyhedron $\mathcal{S}$ that contains all element of $\mathcal{P}$, but no elements from $\mathcal{Q}$.

In a practical data mining project typically many experiments are run and many models are built. It is non-trivial to decide which of them should be used for prediction in the final system. Obviously, if two models achieve the same accuracy on the training set, then it is reasonable to choose the simpler one. The Vapnik–Chervonenkis dimension is a widely accepted model complexity measure in the case of binary classification.

The first chapter of the thesis (Introduction) briefly introduces the field of machine learning and locates convex polyhedron learning in it. Then, without completeness it overviews a set known learning algorithms. The part dealing with collaborative filtering contains novel results too.

The second chapter of the thesis (Algorithms) is about algorithms that use convex polyhedrons for solving various machine learning problems. The first part of the chapter deals with the problem of linear and convex separation. The second part of the chapter gives algorithms for training convex polyhedron classifiers. The third part of the chapter introduces a convex polyhedron based algorithm for collaborative filtering.

The third chapter of the thesis (Model complexity) collects the known facts about the Vapnik–Chervonenkis dimension of convex polyhedron classifiers and proves new results. The fourth chapter (Applications) presents the experiments performed with the proposed algorithms.

Example isn't another way to teach, it is the only way to teach.

Albert Einstein

1 Introduction

Learning from examples is a characteristic ability of human intelligence. For us, learning is as natural as breathing. We not only observe the world, but inherently try to find relationships between our observations. From this point of view, a child learning to ride a bike discovers the connection between his/her perception and traveling safely. A student preparing for an exam tries to understand the connection between the possible questions and the correct answers. In this thesis, *learning* will be considered as discovering the relationship between the features of a phenomenon.

If the features are encoded as numbers, and the relationship between them is discovered by an algorithm, then we talk about *machine learning* (ML) [1]. The input of machine learning is a dataset that was collected by observing the phenomenon. The output is a program that is able to answer certain questions about the phenomenon.

On the map of scientific disciplines machine learning could be placed into the intersection of statistics and computer science. Machine learning aims at inferring from data reliably, therefore it can be viewed as a subfield of statistics. However, machine learning puts great emphasis on computer architectures, data structures, algorithms, and complexity, therefore it can be considered as a subfield of computer science.

One might ask: "Why is it useful if machines learn?" There are plenty of reasons for it:

- In many real world problems it is difficult to formalize the connection between the input and the output, however it is easy to collect corresponding input–output pairs (e.g. face recognition, driving a car). In such cases, ML might be the only way to solve the problem.

- The raw ML solution consists of two simple and automatable steps: collecting data and feeding a learning algorithm with it. Therefore with ML it is possible to get an initial solution quickly. This may significantly reduce development time and cost.

- It happens quite often in engineering practice that the environment of the designed system changes over time. In such cases the adaptiveness of the ML solution is beneficial.

- There are sources that are quickly and continuously producing data (e.g. video cameras, web servers). Often, the data just lies unutilized after storing. ML algorithms may extract valuable information from the available huge amount of unprocessed data.

- ML experiments can help us to understand better how human learning and human intelligence works.

[1] Another popular name of the discipline is data mining.

Now let us introduce the problem more formally. The *phenomenon* is modeled by the random vector $\mathbf{Z}$. The components of $\mathbf{Z}$ are called the *features*. The distribution of $\mathbf{Z}$ denoted by $P_{\mathbf{Z}}$ describes the frequency of encountering particular realizations of $\mathbf{Z}$ in practice.

$P_{\mathbf{Z}}$ is typically unknown, but in some cases one may have some partial knowledge about it (e.g. one might assume that $\mathbf{Z}$ has multinormal distribution). The phenomenon can be either *fully observable* which means that all features are observable, or *partially observable* which means that some features are observable and some are not.

The goal is to estimate $P_{\mathbf{Z}}$ or a well defined part of it, based on a finite sample generated according to $P_{\mathbf{Z}}$. The elements of the sample are called *training example*s, and the sample itself is called the *training set*.

In the rest of this chapter we will overview some special cases of the machine learning problem and investigate a selected subset of known learning algorithms. Furthermore, it will be revealed to the Reader what "convex polyhedron learning" means and why is it useful. I emphasize that the survey about algorithms does *not* want to be exhaustive. The main selection criterion was the degree of connection to the rest of the thesis.

1.1 Classification

In the problem of *classification* the phenomenon is a fully observable 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[2] $g : \mathbb{R}^d \mapsto \mathcal{C}$ called *classifier* such that the probability of error

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

is minimal. Theory says that the minimum of $L(g)$ exists for every distribution of $(\mathbf{X}, Y)$. The best possible classifier is the *Bayes classifier*[3] [Devroye et al., 1996]:

$$g^*(\mathbf{x}) = \arg\max_{y \in \mathcal{C}} \mathbf{P}\{Y = y | \mathbf{X} = \mathbf{x}\}.$$

The minimal probability of error $L^* = L(g^*)$ is called the *Bayes error*. If $L^* = 0$, then the problem is called *separable*, otherwise it is called *inseparable*. In the separable case it is possible to construct a classifier that (almost) never errs. In contrast, in the inseparable case the input does not contain enough information to predict the label without error. Note that in the case of binary classification L^* cannot be larger than 0.5, and $L^* = 0.5$ means that for (almost) every $\mathbf{X}$ the corresponding Y is generated by a coin toss .

Typically, the distribution of $(\mathbf{X}, Y)$ is unknown, so that g^* and L^* is 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

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

[3]The optimum is not unique. Perturbed variants of the Bayes classifier are also optimal, if the probability of perturbation is zero.

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 Bayes classifier g^* on the basis of $\mathbf{T}$. In other words we want to construct a function $g_n : \mathbb{R}^d \times (\mathbb{R}^d \times \mathcal{C})^n \mapsto \mathcal{C}$, called *n-classifier*. This description incorporates the recipe of constructing the classifier from the training set: if we bind all variables except the first d, then we get a classifier. The error of an n-classifier g_n is defined as

$$L(g_n) = \mathbf{P}\{g_n(\mathbf{X}, \mathbf{T}) \neq Y | \mathbf{T}\}.$$

Note that $L(g_n)$ is a random variable because of the random $\mathbf{T}$ in the condition. The quantity $\mathbf{E}\{L(g_n)\}$ is also interesting. This number indicates the quality of the n-classifier on an *average* training set, not *your* training set.

The disadvantage of an n-classifier is that it predefines the number of training examples. It is useful to introduce a related concept that handles arbitrary training set size. A *classification algorithm* is a sequence of functions such that the n-th function is an n-classifier.

A good classification algorithm should produce a good classifier for any distribution of $(\mathbf{X}, Y)$, if the training set is large enough. This requirement can be formalized with the following definition: a classification algorithm $\{g_n\}$ is said to be *universally consistent*, if

$$\lim_{n\to\infty} \mathbf{E}\{L(g_n)\} = L^*$$

with probability one for any distribution of $(\mathbf{X}, Y)$.

Some interesting facts about classification algorithms [Devroye et al., 1996]:

- No universally consistent classification algorithm was known until 1977, when Stone proved that under certain conditions the K *nearest neighbors* algorithm has this property [Stone, 1977].
- For any universally consistent classification algorithm $\{g_n\}$ there exist a distribution of $(\mathbf{X}, Y)$ for which $\mathbf{E}\{L(g_n)\}$ converges to L^* arbitrarily slowly. As a consequence, there is no guarantee that a universally consistent algorithm will perform well in practice.
- For any two n-classifiers g_n and h_n, if $\mathbf{E}\{L(g_n)\} < \mathbf{E}\{L(h_n)\}$ for a distribution of $(\mathbf{X}, Y)$, then there necessarily exists another distribution of $(\mathbf{X}, Y)$ for which $\mathbf{E}\{L(h_n)\} < \mathbf{E}\{L(g_n)\}$. This means that no n-classifier can be inherently superior to any other and there is no best n-classifier.

In the next subsections we will overview a selected subset of known classification algorithms.

1.1.1 Linear classification

Linear classification algorithms are simple, old, and extensively studied. Some of them were already used in 1936, but they are still popular today. They are applied both directly and as component of more complex learning machines.

A set $\mathcal{S} \subset \mathbb{R}^d$ is called a *half-space*, if it can be given in the following form:

$$\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T\mathbf{x} + b \geq 0, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

A binary classifier $g : \mathbb{R}^d \mapsto \{c_1, c_2\}$ is termed *linear*, if $\{\mathbf{x} \in \mathbb{R}^d : g(\mathbf{x}) = c_1\}$ is a half-space. An equivalent definition is the following: a linear classifier is a function $g : \mathbb{R}^d \mapsto \{c_1, c_2\}$ that can be written in the following form:

$$g(\mathbf{x}) = \mathrm{th}(\mathbf{w}^T\mathbf{x} + b), \tag{1.1}$$

where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are the parameters of the classifier, and

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

is the threshold function. Unless otherwise stated we will always assume that $c_1 = 1$ and $c_2 = 0$.

A classification algorithm is called linear, if it produces linear classifiers. The set $\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T\mathbf{x} + b = 0\}$, is called the *decision hyperplane*. The various linear classification algorithms differ in the way of determining $\mathbf{w}$ and b.

1.1.2 Fisher discriminant analysis

Fisher discriminant analysis (FDA) [Fisher, 1936] is probably the oldest recipe for determining $\mathbf{w}$. Its idea is that the scalar product $\mathbf{w}^T\mathbf{x}$ can be viewed as the projection of the input $\mathbf{x}$ to one dimension. Let us introduce the notations

$$n_c = \sum_{i=1}^{n} I\{y_i = c\},$$

$$\mathbf{m}_c = \frac{1}{n_k} \sum_{i=1}^{n} I\{y_i = c\}\mathbf{x_i},$$

$$\mathbf{R}_c = \frac{1}{n_k - 1} \sum_{i=1}^{n} I\{y_i = c\}(\mathbf{x_i} - \mathbf{m_k})(\mathbf{x_i} - \mathbf{m_k})^T$$

for the elementary statistics of the classes ($c = 0, 1$). Then the empirical means and variances of the projected classes can be written as $\mathbf{w}^T\mathbf{m}_c$ and $\mathbf{w}^T\mathbf{R}_c\mathbf{w}$ ($c = 0, 1$). The goal of FDA is to find a vector $\mathbf{w}$ for which

$$\mathcal{F}(\mathbf{w}) = \frac{(\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_0)^2}{\mathbf{w}^T\mathbf{R}_1\mathbf{w} + \mathbf{w}^T\mathbf{R}_0\mathbf{w}} \tag{1.2}$$

the so called Fisher criterion is maximal. In other words, FDA wants to obtain a large between-class variance and a small within-class variance simultaneously. Note that the maximum is not unique, since $\mathcal{F}(\mathbf{w}) = \mathcal{F}(\alpha\mathbf{w})$ for any $\alpha \in \mathbb{R} \setminus \{0\}$.

It can be shown that $\mathcal{F}$ is maximal at[4]

$$\mathbf{w}^* = (\mathbf{R}_1 + \mathbf{R}_0)^{-1}(\mathbf{m}_1 - \mathbf{m}_0). \tag{1.3}$$

It is also true that $\mathbf{w}^*\mathbf{m}_1 \geq \mathbf{w}^*\mathbf{m}_0$, therefore $\mathbf{w}^*$ can be used in (1.1) without flipping the sign.

The original FDA algorithm does not say anything about the offset b. A simple heuristic is setting it to $(\mathbf{w}^*\mathbf{m}_1 - \mathbf{w}^*\mathbf{m}_0)/2$.

1.1.3 Logistic regression

Logistic regression (LOGR) [Wilson and Worcester, 1943] is a classical statistical method that is frequently used in medical and social sciences. It assumes the following interdependency between $\mathbf{X}$ and Y:

$$\mathbf{P}\{Y = 1 | \mathbf{X} = \mathbf{x}\} = \mathrm{sgm}(\mathbf{w}^T\mathbf{x} + b), \tag{1.4}$$

where $\mathrm{sgm}(z) = 1/(1 + \exp(-z))$ is the logistic sigmoid function. When classifying a new input $\mathbf{x}$, logistic regression answers the class with higher probability:

$$g(\mathbf{x}) = \mathrm{th}(\mathbf{P}\{Y = 1 | \mathbf{X} = \mathbf{x}\} - 0.5) = \mathrm{th}(\mathbf{w}^T\mathbf{x} + b).$$

[4] If the inverse exists.

Note that logistic regression is more than a simple "black box": besides classifying the input it also gives the probability of the classes.

In the training phase, the parameters $\mathbf{w}$ and b are calculated via maximum likelihood estimation. This means that $\mathbf{w}$ and b are set such that the conditional probability

$$\mathbf{P}\{Y_i = y_i, \ldots, Y_n = y_n | \mathbf{X}_i = \mathbf{x}_i, \ldots, \mathbf{X}_n = \mathbf{x}_n\} \tag{1.5}$$

is maximal. In other words we want to find the model for which the probability of getting the training labels given the training inputs is maximal. Maximizing (1.5) is equivalent with minimizing

$$\begin{aligned} \mathcal{L}(\mathbf{w}, b) &= -\ln \mathbf{P}\{Y_1 = y_1, \ldots, Y_n = y_n | \mathbf{X}_1 = \mathbf{x}_1, \ldots, \mathbf{X}_n = \mathbf{x}_n\} \\ &= \sum_{i=1}^{n} \left(\ln(1 + \exp(\mathbf{w}^T \mathbf{x}_i + b)) - y_i(\mathbf{w}^T \mathbf{x}_i + b) \right), \end{aligned} \tag{1.6}$$

the negative log-likelihood.

$\mathcal{L}$ is convex, therefore its minimum can be approximated well by iterative optimization algorithms (e.g. gradient descent, Newton's method).

1.1.4 Artificial neuron

The linear classifier $g(\mathbf{x}) = \mathrm{th}(\mathbf{w}^T \mathbf{x} + b)$ can also be viewed as a simple model of the biological neuron [McCullogh and Pitts, 1943]. In this interpretation the elements of $\mathbf{w}$ are input connection strengths. Stimulating the neuron with input $\mathbf{x}$ causes an activation $\mathbf{w}^T \mathbf{x}$ in the neuron. If the activation level is greater than b, then neuron "fires" and emits a signal on its output.

It is a natural idea that training should be done by minimizing

$$\mathcal{N}(\mathbf{w}, b) = \sum_{i=1}^{n} I\{\mathrm{th}(\mathbf{w}^T \mathbf{x}_i + b) \neq y_i\},$$

the number of misclassifications in the training set.

Unfortunately, the minimization of $\mathcal{N}$ is difficult, because the functions I and th are not differentiable. A straightforward way to overcome this difficulty is replacing them by smooth functions. The replacements will assume that the class labels are $c_1 = 1$ and $c_2 = 0$.

If $I\{\alpha \neq \beta\}$ is replaced by $-\ln(\alpha^\beta (1-\alpha)^{1-\beta})$ and $\mathrm{th}(\gamma)$ by $\mathrm{sgm}(\gamma)$, then we get logistic regression. However, this is not the only possible choice.

Smooth perceptron

If $I\{\alpha \neq \beta\}$ is replaced by $(\alpha - \beta)^2$ and $\mathrm{th}(\gamma)$ by $\mathrm{sgm}(\gamma)$, then we get the smooth variant of Rosenblatt's perceptron [Rosenblatt, 1962], referred as *smooth perceptron* (SPER). In this case the function to minimize is the following:

$$\mathcal{P}(\mathbf{w}, b) = \sum_{i=1}^{n} (\mathrm{sgm}(\mathbf{w}^T \mathbf{x}_i + b) - y_i)^2.$$

Finding the global minimum of $\mathcal{P}$ is difficult, because $\mathcal{P}$ is nonconvex. However, a local minimum can be computed easily with iterative methods, and this is often sufficient in practice.

Adaptive linear neuron

If $I\{\alpha \neq \beta\}$ is replaced by $(\alpha - \beta)^2$ and $\text{th}(\gamma)$ by $\gamma + 0.5$, then we get the *adaptive linear neuron* (ALN) [Widrow, 1960]. Now the function to minimize is the following:

$$\mathcal{A}(\mathbf{w}, b) = \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i + b + 0.5 - y_i)^2.$$

$\mathcal{A}$ is convex and quadratic, therefore the minimization can be done efficiently. A disadvantage is that $\mathcal{A}$ is quite different from the original function $\mathcal{N}$. As a consequence, ALN classifiers tend to be less accurate than other linear classifiers.

1.1.5 Linear support vector machine

Support vector machine (SVM) [Boser et al., 1992] is quite a new invention in machine learning. The linear variant of SVM (LSVM) is a linear classification algorithm. The goal of LSVM to separate the classes from each other such that the distance between the decision hyperplane and the closest training examples (called *margin*) is maximal. Assuming class labels $c_1 = +1$ and $c_2 = -1$ the requirement can be formalized as the following quadratic programming problem:

$$\begin{aligned} &\text{variables: } \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R},\ \boldsymbol{\xi} \in \mathbb{R}^n \\ &\text{minimize: } \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to: } (\mathbf{w}^T\mathbf{x}_i + b)y_i \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n. \end{aligned} \tag{1.7}$$

The role of variable $\boldsymbol{\xi}$ is to make the problem feasible for every training set. The parameter C can be used as a tradeoff between training set classification accuracy and maximizing the margin. For solving (1.7) one can use a general quadratic programming solver or a specialized algorithm like sequential minimal optimization (SMO) [Platt, 1999].

1.1.6 Nonlinear classification

In many real world training sets, the classes cannot be separated from each other with a hyperplane. One possible solution is to consider this as an effect of noise and still apply a linear classifier. A different and often more accurate approach is to apply a nonlinear classifier. In the following subsections we will overview a very limited subset of known nonlinear classification algorithms.

1.1.7 K nearest neighbors

The K *nearest neighbors* (KNN) [Fix and Hodges, 1951] approach is based on the assumption that if two inputs are similar, then their labels are probably identical. Let $\delta : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ be a distance function. In the training phase KNN just memorizes the training examples. Then a new input $\mathbf{x}$ is classified as follows:

1. Calculate the distance between $\mathbf{x}$ and all training inputs with respect to δ.
2. Determine the indices of the K closest training inputs to $\mathbf{x}$, and denote them by $i_1, i_2, \ldots, i_K$.
3. Return the most frequent label (or one of the most frequent labels) from $\{y_{i_1}, y_{i_2}, \ldots, y_{i_K}\}$.

An appealing property of KNN classifiers is that they provide a nice explanation to their decision (e.g. "Joe was classified as a beer lover, because the most similar person in the database, Tom, is also one."). The weak point of most KNN implementations is limited scalability (because one has to iterate over the training examples to classify an input).

1.1.8 ID3 decision tree

The key idea of decision tree algorithms is "divide and conquer". The outline of the approach is the following:

- In the training phase, the training set is partitioned by applying a *splitting rule* recursively.
- In the classification phase, the partition of the input is determined, and the most frequent label (or one of the most frequent labels) in the selected partition is returned.

Here we overview a simple decision tree variant called *iterative dichotomizer 3* (ID3) [Quinlan, 1986]. At first let us assume that all features are categorical. The entropy of a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ denoted by $H(\mathcal{D})$ is defined as

$$p_k = \frac{1}{n + M\beta}\left(\beta + \sum_{i=1}^{n} I\{y_i = c_k\}\right), \quad k = 1, \ldots, M,$$
$$H(\mathcal{D}) = -\sum_{k=1}^{M} p_k \log_2(p_k),$$

where $\beta > 0$ is called the Laplace smoothing term.

Assume that the possible values of the j-th feature are $v_1, \ldots, v_N$. Splitting $\mathcal{D}$ along the j-th feature means that the following sub datasets are created:

$$\mathcal{D}_l = \{(\mathbf{x}, y) \in \mathcal{D} : x_j = v_l\}, \quad l = 1, \ldots, N.$$

The information gain of the split is defined as

$$\mathcal{G} = H(\mathcal{D}) - \sum_{l=1}^{N} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} H(\mathcal{D}_l). \tag{1.8}$$

Originally, ID3 was designed for categorical features, but it can be extended to handle continuous features too. Splitting dataset $\mathcal{D}$ along continuous feature j using value α results the following sub datasets:

$$\mathcal{D}_1(\alpha) = \{(\mathbf{x}, y) \in \mathcal{D} : x_j \leq \alpha\},$$
$$\mathcal{D}_2(\alpha) = \{(\mathbf{x}, y) \in \mathcal{D} : x_j > \alpha\}.$$

The information gain of the split is defined as

$$\mathcal{G} = \max_{\alpha \in \mathbb{R}} H(\mathcal{D}) - \sum_{l=1}^{2} \frac{|\mathcal{D}_l(\alpha)|}{|\mathcal{D}|} H(\mathcal{D}_l(\alpha)). \tag{1.9}$$

In practice it is often too expensive to try every possible value of α. A simple solution is to consider only K values, $\alpha_1, \ldots, \alpha_K$ chosen so that the partitions $\{(\mathbf{x}, y) \in \mathcal{D} : \alpha_k < x_j \leq \alpha_{k+1}\}$ are (nearly) equally sized.

The ID3 rule splits the dataset along the feature that gives the highest information gain. ID3 training applies this rule recursively until the information gain is not lower than a predefined limit $\mathcal{G}_{\min}$. The split features (and α values) found by the algorithm can be stored in a tree structure such that each node corresponds to a sub dataset. The class frequencies of the sub datasets can also be stored in the tree.

Using this tree structure, the classification of a new input can be done in $O(L)$ time, where L is the depth of the tree. Note that the time requirement does not depend on the number of features d, and only very elementary operations are needed (array indexing, scalar comparison). Therefore, ID3 classifiers can be faster than even linear classifiers in the classification phase.

Another strong point of the ID3 is user friendly explanation generation. The path to the selected leaf node can be viewed as a conjunction of simple statements (e.g. "you will probably like this French restaurant, because you like wine and your favorite city is Paris.") A disadvantage of ID3 is that it tends to be inaccurate on problems with continuous features.

1.1.9 Multilayer perceptron

Multilayer perceptron (MLP) is one of the most popular *artificial neural networks* [Haykin, 2008]. An MLP consists of simple processing units called neurons that are arranged in layers. The first layer is called input and the last is called output layer. The layers between them are termed hidden layers.

Two neurons are connected if and only if they are in consecutive layers. A weight is associated with each connection. The neurons of the input layer contain the identity function. Every other neuron consists of a linear combiner and a nonlinear activation function.

Assuming one hidden layer and logistic activation function, the answer of the network to input $\mathbf{x}$ denoted by $g(\mathbf{x})$ is the following:

$$h_k = \operatorname{sgm}\left(b_k + \sum_{j=1}^{d} w_{jk} x_j\right), \quad k = 1, \ldots, K, \tag{1.10}$$

$$g(\mathbf{x}) = \operatorname{th}\left(\operatorname{sgm}\left(c + \sum_{k=1}^{K} h_k v_k\right) - 0.5\right),$$

where $w_{jk}, v_k \in \mathbb{R}$ called weights and $b_k, c \in \mathbb{R}$ called biases are the parameters of the model ($j = 1, \ldots, d$, $k = 1, \ldots, K$). The structure of this network is shown in Figure (1.1).

Denote the matrix of w_{jk} values by $\mathbf{W}$, the vector of b_k values by $\mathbf{b}$, and the vector of v_k values by $\mathbf{v}$. Training can be done by minimizing

$$\mathcal{M}(\mathbf{W}, \mathbf{b}, \mathbf{v}, c) = \sum_{i=1}^{n} \left(y_i - \sum_{i=1}^{n} \operatorname{sgm}\left(c + \sum_{k=1}^{K} \operatorname{sgm}\left(b_k + \sum_{j=1}^{d} w_{jk} x_{ij}\right) v_k\right)\right)^2,$$

the sum of squared errors between the label and the raw output of the network.

A local minimum of $\mathcal{M}$ can be found with the backpropagation algorithm [Werbos, 1974] which is an efficient implementation of gradient descent for this particular objective function.

1.1.10 Support vector machine

The nonlinear *support vector machine* (SVM) [Boser et al., 1992] can be obtained from linear SVM by rewriting the original optimization problem and replacing the scalar product with a

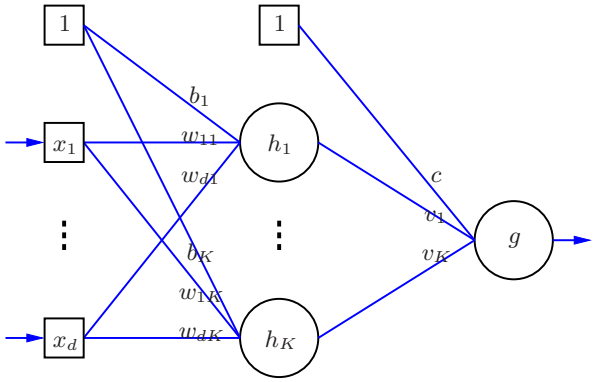


Figure 1.1: The structure of an MLP with one hidden layer.

kernel function $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. This learning machine has nice theoretical properties and it often shows outstanding performance in practice, therefore it has become very popular recently. Here I only give a *very* brief overview of SVM. Those who are interested can find more details e.g. in [Burges, 1998].

Let us assume class labels $c_1 = +1$ and $c_2 = -1$. The answer of the SVM classifier for input $\mathbf{x}$ is the following[5]:

$$g(\mathbf{x}) = \mathrm{th}\left(\sum_{i=1}^{n} \alpha_i y_i \mathcal{K}(\mathbf{x}, \mathbf{x}_i)\right). \tag{1.11}$$

Training examples for which the corresponding α_i is not zero are called support vectors. The training procedure consists of solving the following constrained optimization problem:

$$\begin{aligned} &\text{variables: } \boldsymbol{\alpha} \in \mathbb{R}^n \\ &\text{maximize: } \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{subject to: } 0 \le \alpha_i \le C, \quad i = 1, \ldots, n. \end{aligned} \tag{1.12}$$

Two common choices for $\mathcal{K}$ are:

- Polynomial kernel: $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^p$,
- Gaussian kernel: $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$.

In both cases, the objective function of (1.12) is convex quadratic, therefore the problem to solve is a quadratic program. Similarly to linear SVM, for computing the solution one can use a general quadratic programming solver or a specialized algorithm like SMO [Platt, 1999].

Most algorithms appearing in this thesis are so simple that they can be easily implemented from scratch, but this is not true for SVM (and linear SVM). In most cases it is worthwhile to use an existing fine-tuned SVM implementation like svm-light [Joachims, 1999] or libsvm [Chang and Lin, 2001].

[5]It is possible to also include a bias term in the model. Here the bias was omitted for simplicity.

1.1.11 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

$$\begin{aligned} g(\mathbf{x}) =& \text{th}(\min\{\mathbf{w}_1^T\mathbf{x} + b_1, \ldots, \mathbf{w}_K^T\mathbf{x} + b_K\}) \\ =& \text{th}(-\max\{-\mathbf{w}_1^T\mathbf{x} - b_1, \ldots, -\mathbf{w}_K^T\mathbf{x} - b_K\}), \end{aligned} \tag{1.13}$$

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

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.

1.2 Regression

In the problem of *regression* the phenomenon is a fully observable 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. Some commonly used alternative error measures are the following:

- Mean absolute error: $\mathbf{E}\{|g(\mathbf{X}) - Y)|\}$,
- Mean percentage error: $\mathbf{E}\{|g(\mathbf{X})/Y - 1| \cdot 100\}$.

It is true again that the minimum of $L(g)$ exists for every distribution of $(\mathbf{X}, Y)$. The best possible predictor is the *regression function*[6] [Györfi et al., 2002]:

$$g^*(\mathbf{x}) = \mathbf{E}\{Y|\mathbf{X} = \mathbf{x}\}.$$

The minimal mean squared error $L^* = L(g^*)$ is called the *noise level*.

[6]The optimum is not unique. Perturbed versions of the regression function are also optimal, if the probability of perturbation is zero.

Analogously with classification, we can introduce the concepts n-predictor and regression algorithm. An *n-predictor* is a function that maps from $\mathbb{R}^d \times (\mathbb{R}^d \times \mathcal{C})^n$ to $\mathbb{R}$. A *regression algorithm* is a sequence of functions such that the n-th function is an n-predictor. The error of an n-predictor g_n is defined as

$$L(g_n) = \mathbf{E}\{(g_n(\mathbf{X}, \mathbf{T}) - Y)^2 | \mathbf{T}\}.$$

A regression algorithm $\{g_n\}$ is said to be *universally consistent*, if

$$\lim_{n\to\infty} \mathbf{E}\{L(g_n)\} = L^*$$

with probability one for all distributions of $(\mathbf{X}, Y)$.

1.2.1 Linear regression

Linear regression (LINR) is probably the oldest machine learning technique. According to [Pearson, 1930], the first regression line was plotted at a lecture by Galton in 1877. The rigorous description of the algorithm appeared first in [Pearson, 1896].

A predictor $g : \mathbb{R}^d \mapsto \mathbb{R}$ is called *linear* if it can be written in the following form:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b. \tag{1.14}$$

The goal of linear regression is to minimize

$$\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x_i} + b - y_i)^2,$$

the sum of squared errors on the training set. With differentiation it can be shown that the minimum of $\mathcal{L}$ is located at[7]

$$\begin{pmatrix} w_1^* \\ \vdots \\ w_d^* \\ \hline b^* \end{pmatrix} = \left(\begin{array}{ccc|c} \sum_{i=1}^n x_{i1}x_{i1} & \cdots & \sum_{i=1}^n x_{id}x_{i1} & \sum_{i=1}^n x_{i1} \\ \vdots & \ddots & \vdots & \vdots \\ \sum_{i=1}^n x_{i1}x_{id} & \cdots & \sum_{i=1}^n x_{id}x_{id} & \sum_{i=1}^n x_{id} \\ \hline \sum_{i=1}^n x_{i1} & \cdots & \sum_{i=1}^n x_{id} & \sum_{i=1}^n 1 \end{array}\right)^{-1} \left(\begin{array}{c} \sum_{i=1}^n x_{i1}y_i \\ \vdots \\ \sum_{i=1}^n x_{id}y_i \\ \hline \sum_{i=1}^n y_i \end{array}\right) \tag{1.15}$$

1.2.2 Nonlinear regression

All of the discussed nonlinear classification algorithms (KNN, ID3, MLP, SVM) can be used as nonlinear regression techniques after some modifications:

- KNN: at prediction return the average of the neighbors' target value.
- ID3: at prediction return the average target value of the partition, at training use variance instead of entropy.
- MLP: omit the sigmoid function from the output neuron, and omit the threshold function from the prediction formula.
- SVM: omit the threshold function, and replace the per example loss function to the ϵ-insensitive loss (see [Burges, 1998]).

[7] If the inverse exists.

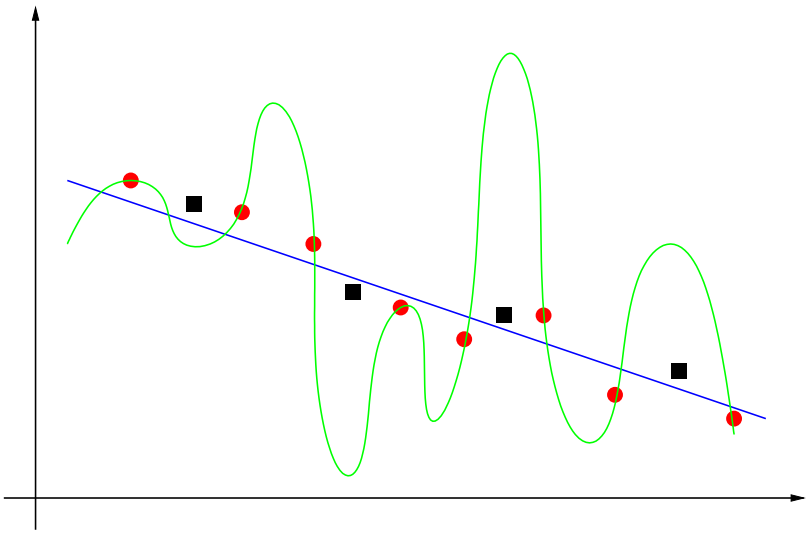


Figure 1.2: The red dots represent training examples and the black squares new examples in a regression problem. The green curve and the blue line represent two predictors. The green predictor fits perfectly to the training examples, but the blue one generalizes better.

Convex polyhedron 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:

$$\begin{aligned} g(\mathbf{x}) &= \min\{\mathbf{w}_1^T\mathbf{x} + b_1, \ldots, \mathbf{w}_K^T\mathbf{x} + b_K\} \\ &= -\max\{-\mathbf{w}_1^T\mathbf{x} - b_1, \ldots, -\mathbf{w}_K^T\mathbf{x} - b_K\}. \end{aligned} \tag{1.16}$$

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 $\mathbb{R}^{d+1}$.

1.3 Techniques against overfitting

If a learning machine performs well on the training set but poorly (or not so well) on new examples, then we talk about *overfitting*. Figure (1.2) illustrates the phenomenon in the case of regression.

Overfitting is a natural consequence of using a finite training set, therefore it is an issue in almost every machine learning project. The question is not how to eliminate it completely but how to handle it well. Fighting against overfitting to aggressively may lead to *underfitting*, which means that the learning machine performs poorly both on training and new examples.

As we have seen previously, the training procedure of a learning machine often means the unconstrained minimization of a differentiable objective function. In this case some usual techniques against overfitting are the following:

- L2 regularization: The term $\frac{1}{2}\lambda||\mathbf{p}||_{L2}^2 = \frac{1}{2}\lambda\sum_{k=1}^{K} p_k^2$ is added to the objective function, where the vector $\mathbf{p} = (p_1, \ldots, p_K)$ contains a subset of the model's parameters and the

scalar λ is called the regularization coefficient. Introducing this term penalizes the large magnitude of model parameters.

- L1 regularization: The term $\lambda||\mathbf{p}||_{L1} = \lambda \sum_{k=1}^{K} |p_k|$ is added to the objective function. Introducing this term may cause that the optimal value of some parameters will be exactly zero.
- Early stopping: The iterative minimization algorithm is stopped before reaching a stationary point of the loss function.
- It is often useful to apply both regularization and early stopping.

Without completeness, some other common techniques against overfitting are the following:

- In the case of FDA a convenient way to decrease overfitting is to add $\lambda \mathbf{w}^T \mathbf{w}$ to the denominator of the objective function.
- In the case of KNN overfitting can be decreased by increasing the number of relevant neighbors K.
- In the case of ID3 overfitting can be decreased by increasing the Laplace smoothing term β or the information gain threshold $\mathcal{G}_{\min}$.
- In the case of SVM and LSVM overfitting can be decreased by decreasing the tradeoff parameter C.

1.4 Collaborative filtering

In *collaborative filtering*, the phenomenon is a fully observable 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 $\mathbb{R}$ 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 $g : \{1, \ldots, N_U\} \times \{1, \ldots, N_I\} \mapsto \mathbb{R}$ such that mean squared error

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

is minimal. The most commonly used alternative error measure is $\mathbf{E}\{|g(U, I) - R|\}$, the mean absolute error.

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

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

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.

When we predict a given rating r_{ui}, we refer to the user u as *active user* and to the item i as *active item*. The (u,i) pair of active user and active item is termed *query*. The set of items rated by user u is denoted by $\mathcal{T}_u = \{i : \exists u : (u,i) \in \mathcal{T}\}$. The set of users who rated item i is denoted by $\mathcal{T}^{(i)} = \{u : \exists i : (u,i) \in \mathcal{T}\}$.

1.4.1 Double centering

Double centering (DC) is a very basic approach that gives a rough solution to the problem. The answer of DC for user u and item i is

$$g(u,i) = b_u + c_i,$$

where $b_1, \ldots, b_{N_U}$ called user biases and $c_1, \ldots, c_{N_I}$ called item biases are the parameters of the model. Training can be done by minimizing

$$\mathcal{D}(\mathbf{b},\mathbf{c}) = \sum_{(u,i)\in\mathcal{T}} ((b_u + c_i) - r_{ui})^2,$$

the sum of squared errors on the training set.

$\mathcal{D}$ is convex and quadratic therefore its minimum can be expressed in closed form. The difficulty is that typically $\mathcal{D}$ has so many variables that computing the closed form solution is intractable. A faster method that produces an approximate solution is the following:

1. Initialize $\mathbf{b}$ to $\mathbf{0}$.
2. Repeat E times:
   - Compute the $\mathbf{c}$ that minimizes $\mathcal{D}$ for fixed $\mathbf{b}$:
     $c_i \leftarrow \sum_{u\in\mathcal{T}^{(i)}} (r_{ui} - b_u)/|\mathcal{T}^{(i)}|$, for $i = 1, \ldots, N_I$.
   - Compute the $\mathbf{b}$ that minimizes $\mathcal{D}$ for fixed $\mathbf{c}$:
     $b_u \leftarrow \sum_{i\in\mathcal{T}_u} (r_{ui} - c_i)/|\mathcal{T}_u|$, for $u = 1, \ldots, N_U$.

Thus, we are alternating between minimizing the objective function in $\mathbf{c}$ and in $\mathbf{b}$. Note that generally there is no guarantee for the speed of convergence. A possible tool for reducing the number of iterations is the Hooke–Jeeves pattern search algorithm [Hooke and Jeeves, 1961]. I also mention that an alternative approach for the fast approximate minimization of $\mathcal{D}$ is the conjugate gradient method [Shewchuk, 1994].

1.4.2 Matrix factorization

The idea behind *matrix factorization* (MF) techniques is simple: we approximate the rating matrix $\mathbf{R}$ as the product of two matrices:

$$\mathbf{R} \approx \mathbf{PQ},$$

where $\mathbf{P}$ is an $N_U \times L$ and $\mathbf{Q}$ is an $L \times N_I$ matrix. We call $\mathbf{P}$ the user factor matrix and $\mathbf{Q}$ the item factor matrix, and L is the rank of the given factorization.

The prediction for user u and item i is the (u,i)-th element of $\mathbf{PQ}$:

$$g(u,i) = \sum_{l=1}^{L} p_{ul} q_{li}.$$

The training can be done by minimizing

$$\mathcal{M}(\mathbf{P},\mathbf{Q}) = \sum_{(u,i)\in\mathcal{T}} \left(\left(\sum_{l=1}^{L} p_{ul} q_{li} \right) - r_{ui} \right)^2,$$

the sum of squared errors at the known positions of the rating matrix.

$\mathcal{M}$ is a polynomial of degree 4. It is convex in $\mathbf{P}$ and convex in $\mathbf{Q}$, but it is not necessarily convex in $(\mathbf{P},\mathbf{Q})$. The approximate minimization of $\mathcal{M}$ can be done e.g. by stochastic gradient descent [Takács et al., 2007] or alternating least squares [Bell and Koren, 2007].

1.4.3 BRISMF

In this section, I propose a practical and efficient variant of MF, called *biased regularized incremental simultaneous matrix factorization* (BRISMF) [Takács et al., 2009]. The prediction of BRISMF for user u and item i is

$$g(u,i) = b_u + c_i + \sum_{l=1}^{L} p_{ul} q_{li},$$

Contrary to basic MF, now the model contains user biases $b_1 \dots b_{N_U}$ and item biases $c_1 \dots c_{N_I}$ too. The function to minimize is

$$\mathcal{M}(\mathbf{P},\mathbf{Q},\mathbf{b},\mathbf{c}) = \sum_{(u,i)\in\mathcal{T}} \left((g(u,i) - r_{ui})^2 + \lambda_U \sum_{l=1}^{L} p_{ul}^2 + \lambda_I \sum_{l=1}^{L} q_{li}^2 \right).$$

The difference from basic MF is that now the objective function contains regularization terms too (λ_U and λ_I are called user and item regularization coefficients). The minimization of $\mathcal{M}$ is done by stochastic (aka incremental) gradient descent. The pseudo-code of the training algorithm can be seen in Figure 1.3.

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

- $R \in \mathbb{R}$: range of random number generation at model initialization,
- $E \in \mathbb{N}$: number of epochs (iterations over the training set),
- $\eta_U, \eta_I \in \mathbb{R}$: user and item learning rates[8] — they control the step size at model update,
- $\lambda_U, \lambda_I \in \mathbb{R}$: user and item regularization coefficients — they control how aggressively the factors are pushed towards zero,
- $D \in \{0,1\}$: ordering flag — controls whether ordering by date should be used within users.

[8] The idea of using different meta-parameters for users and items was suggested by my colleague, István Pilászy.

Input: $r_{ui} : (u, i) \in \mathcal{T}$, $|\mathcal{T}| = n$ // the training set
Input: R, E, η_U, η_I, λ_U, λ_I, D // meta-parameters
Output: $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$, $\mathbf{c}$ // the trained model

$\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$, $\mathbf{c} \leftarrow$ uniform random numbers from $[-R, R]$ // initialization

for $e \leftarrow 1$ **to** E **do** // for all epochs
  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
      $a \leftarrow b_u + c_i + \sum_{l=1}^{L} p_{ul} q_{li}$ // 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
        $p \leftarrow p_{ul}$ // save current value
        $p_{ul} \leftarrow p_{ul} - \eta_U(\varepsilon q_{li} + \lambda_U p_{ul})$
        $q_{li} \leftarrow q_{li} - \eta_I(\varepsilon p + \lambda_I q_{li})$
      end
    end
  end
end

Figure 1.3: Training algorithm for BRISMF.

In the time dependent version of collaborative filtering a time attribute is also given for each rating, and a task is to predict future ratings from past ones. The setting $D = 1$ makes the algorithm able to deal with this variant of the problem in a simple and computationally efficient way.

It is important that $\mathbf{P}$ and $\mathbf{Q}$ are initialized randomly. For example, if they were initialized with zeros, then they would not change during the training. The time requirement of the algorithm is $O(EL|\mathcal{T}|)$, therefore it is able to deal with very large datasets.

BRISMF differs from other MF techniques in a few important aspects. Here is a summary of differences from the most common alternatives:

- Simon Funk's MF [Funk, 2006]: Simon Funk's approach applies a sequence of rank 1 approximations instead of updating all factors simultaneously. It is not specified when the learning procedure has to step to the next factor. His approach converges slower than BRISMF, because it iterates over $\mathbf{R}$ more times. Moreover, Simon Funk's MF does not contain bias terms.
- Paterek's MF [Paterek, 2007]: The idea of user and item bias appeared at the same time in Paterek's work (in section "Improved regularized SVD") and in [Takács et al., 2007] (in section "Constant values in matrices"). Paterek's MF variant shares some common features with BRISMF, but it uses Simon Funk's approach to update factors.
- BellKor's MF [Bell and Koren, 2007]: BellKor's MF does not contain bias terms, and it uses alternating least squares for the approximate minimization of the objective function. This means that $\mathbf{P}$ and $\mathbf{Q}$ are initialized randomly, one of them is recomputed using a least squares solver while the other is constant, then the other is recomputed, and these two alternating steps are performed E times. The time requirement of alternating least squares is $O(E(N_U + N_I)L^3 + EL^2|\mathcal{T}|)$, and this upper bound is close to the true computational complexity. Therefore, BellKor's MF is less scalable than BRISMF.
- None of the previous three MF variants apply ordering by date within user ratings.

1.4.4 Neighbor based methods

Neighbor based methods exploit the observation that similar users rate similar items similarly. In the item neighbor based version of the approach, the prediction formula contains the similarities between the active item and other items rated by the active user.

Here I present an elegant and efficient item neighbor based method (referred as NSVD1) that infers the similarity measure from the data. The earliest variant of the approach appeared in Paterek's pioneering work [Paterek, 2007].

The answer of a basic neighbor based method for user u and item i is

$$g(u,i) = b_u + c_i + \frac{1}{\sqrt{|\mathcal{T}_u|}} \sum_{j \in \mathcal{T}_u} s_{ji},$$

where $b_1, \ldots, b_{N_U}$ and $c_1, \ldots, c_{N_I}$ are user and item biases as usual, and $\mathcal{T}_u$ is the set of items rated by user u. The $s_{ji} \in \mathbb{R}$ values can be interpreted as similarities between items in a sense: $s_{ji} > 0$ ($s_{ji} < 0$) means that if a user has rated item j, than he/she will probably like (dislike) item i more than an average user, and $s_{ji} = 0$ means that there is no such connection between items j and i.

If all s_{ji} values are 0 in the sum, then the prediction will be $b_u + c_i$, therefore $b_u + c_i$ can be considered as the default answer of the model. The role of the normalization factor $1/\sqrt{|\mathcal{T}_u|}$ is

to control the deviation of the output around $b_u + c_i$. If we used $1/|\mathcal{T}_u|^0 = 1$ instead, then it would be difficult to keep the answer in a reasonable range. If we used $1/|\mathcal{T}_u|$, then the model would be too conservative for users with many ratings.

The matrix of s_{ji} values denoted by $\mathbf{S}$ can be called the item–item similarity matrix (j is the row and i is the column index). It is possible to consider $\mathbf{S}$ as the parameter of the model, and do the training via stochastic gradient descent [Paterek, 2007]. However, this approach is often inefficient, since $\mathbf{S}$ has N_I^2 elements, and in a typical collaborative filtering setting N_I is large.

The key idea of NSVD1 is to approximate the similarity matrix as $\mathbf{S} \approx \mathbf{WQ}$, the product of two lower-rank matrices. We call $\mathbf{Q} \in \mathbb{R}^{L \times N_I}$ as the primary item factor matrix and $\mathbf{W} \in \mathbb{R}^{N_I \times L}$ the secondary item factor matrix.

If we introduce the the notation

$$p_{ul} = \left(\frac{1}{\sqrt{|\mathcal{T}_u|}} \sum_{j \in \mathcal{T}_u} w_{jl} \right),$$

then the answer of NSVD1 for user u and item i can be written as

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

Thus, the prediction formula is the same as in the case of BRISMF. The difference is that now the p_{ul} values are not parameters of the model. They are instead functions of the secondary item factors. From now we will refer to the p_{ul} values as virtual user factors.

Training can be done by minimizing

$$\mathcal{N}(\mathbf{Q}, \mathbf{W}, \mathbf{b}, \mathbf{c}) = \sum_{(u,i) \in \mathcal{T}} \left((g(u,i) - r_{ui})^2 + \lambda_U \sum_{l=1}^{L} p_{ul}^2 + \lambda_I \sum_{l=1}^{L} q_{li}^2 \right),$$

the regularized sum of squared errors on the training examples.

For the approximate minimization of $\mathcal{N}$, there exist different variants of stochastic gradient descent. Figure 1.4 presents the pseudo-code of my proposed variant that was introduced in [Takács et al., 2008].

The meta-parameters of the algorithm are the same as in the case of BRISMF. Note that the naive implementation of stochastic gradient descent would be inefficient, since it would iterate over all ratings of the given user at each training example. The presented algorithm overcomes this difficulty by processing the training examples user-wise, and iterating over the ratings of each user three times:

- In the first iteration, the virtual user factors are computed.
- In the second iteration, the virtual user factors and the primary item factors are updated.
- In the third iteration, the change of the virtual user factors is distributed among the secondary item factors.

The derivation of the update formula for p_{ul} is the following: If we differentiate the (u,i)-th term of $\mathcal{N}$ with respect to w_{jl}, then we get

$$2 \frac{1}{\sqrt{|\mathcal{T}_u|}} (q_{li} + \lambda_U p_{ul}).$$

Input: $r_{ui} : (u, i) \in \mathcal{T}$, $|\mathcal{T}| = n$ // the training set
Input: R, E, η_U, η_I, λ_U, λ_I // meta-parameters
Output: $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$, $\mathbf{c}$ // the trained model

$\mathbf{P}$, $\mathbf{Q}$, $\mathbf{b}$, $\mathbf{c} \leftarrow$ uniform random numbers from $[-R, R]$ // initialization

for $e \leftarrow 1$ **to** E **do** // for all epochs
  for $u \leftarrow 1$ **to** N_U **do** // for all users
    $\mathcal{T}_u \leftarrow \{i : \exists u : (u, i) \in \mathcal{T}\}$ // set of rated items
    $\mathcal{I} \leftarrow$ 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
    $p_{u1}, \ldots, p_{uL}, \leftarrow$ zeros // initialize virtual user factors
    for i **in** $\mathcal{I}$ **do** // calculate virtual user factors
      for $l \leftarrow 1$ **to** L **do**
        $p_{ul} \leftarrow p_{ul} + w_{il}/|\sqrt{\mathcal{T}_u}|$
        $p_{ul}^{\text{old}} \leftarrow p_{ul}$
      end
    end
    for i **in** $\mathcal{I}$ **do**
      $a \leftarrow b_u + c_i + \sum_{l=1}^{L} p_{ul} q_{li}$ // calculate answer
      $\varepsilon \leftarrow a - y_i$ // calculate error
      for $l \leftarrow 1$ **to** L **do**
        $p \leftarrow p_{ul}$
        $p_{ul} \leftarrow p_{ul} - \eta_U(\varepsilon q_{li} + \lambda_U p_{ul})$ // update virtual user factor
        $q_{li} \leftarrow q_{li} - \eta_I(\varepsilon p + \lambda_I q_{li})$ // update primary item factor
      end
    end
    for i **in** $\mathcal{I}$ **do** // update secondary item factors
      for $l \leftarrow 1$ **to** L **do**
        $w_{il} \leftarrow w_{il} + r_{ui}(p_{ul} - p_{ul}^{\text{old}})/\sqrt{|\mathrm{T}_u|}$
      end
    end
  end
end

Figure 1.4: Training algorithm for NSVD1.

Thus, the change of w_{jl} after making a step into the direction the negative gradient is

$$\Delta w_{jl} = -\eta_U \frac{1}{\sqrt{|\mathcal{T}_u|}}(q_{li} + \lambda_U p_{ul}).$$

Consequently, the update formula for p_{ul} is

$$p_{ul} \leftarrow p_{ul} + \left(\frac{1}{\sqrt{|\mathcal{T}_u|}} \sum_{j \in \mathcal{T}_u} r_{uj} \Delta w_{jl} \right) = p_{ul} + \eta_U \left(q_l i + \lambda_U p_{ul} \right) \frac{\sum_{j \in \mathcal{T}_u} 1}{|\mathcal{T}_u|}.$$

1.4.5 Convex polyhedron methods

With the generalization of matrix factorization it is possible introduce convex polyhedron models for collaborative filtering. Assume that we have K user factor matrices $\mathbf{P}^{(1)}, \ldots, \mathbf{P}^{(K)}$, an item factor matrix $\mathbf{Q}$, a user bias vector $\mathbf{b}$ and an item bias vector $\mathbf{c}$. The answer of the convex polyhedron predictor for user u and item i can be defined as

$$\begin{aligned} g(u,i) &= b_u + c_i + \min \left\{ \left(\sum_{l=1}^{L} p_{ul}^{(1)} q_{li} \right), \ldots, \left(\sum_{l=1}^{L} p_{ul}^{(K)} q_{li} \right) \right\} \\ &= b_u + c_i - \max \left\{ - \left(\sum_{l=1}^{L} p_{ul}^{(1)} q_{li} \right), \ldots, - \left(\sum_{l=1}^{L} p_{ul}^{(K)} q_{li} \right) \right\}. \end{aligned} \tag{1.17}$$

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

$$\begin{aligned} g(u,i) &= b_u + c_i + \min \left\{ \left(\sum_{l=1}^{L} p_{ul} q_{li}^{(1)} \right), \ldots, \left(\sum_{l=1}^{L} p_{ul} q_{li}^{(K)} \right) \right\} \\ &= b_u + c_i - \max \left\{ - \left(\sum_{l=1}^{L} p_{ul} q_{li}^{(1)} \right), \ldots, - \left(\sum_{l=1}^{L} p_{ul} q_{li}^{(K)} \right) \right\}. \end{aligned} \tag{1.18}$$

Training such models is not easy because of the non differentiable minimum (maximum) function appearing in the formulae.

1.5 Other machine learning problems

The rest of the thesis will focus on convex polyhedron algorithms for classification and collaborative filtering. Having said this, here I briefly sketch some other machine learning problems too, in order to demonstrate the richness of the discipline.

1.5.1 Clustering

In the problem of clustering the phenomenon is a partially observable pair $(\mathbf{X}, Y)$, where

- the observable $\mathbf{X}$ taking values from $\mathbb{R}^d$ is called *input*, and
- the unobservable Y taking values from $\mathcal{C} = \{c_1, \ldots, c_M\}, M \geq 2$ is called *label*.

The goal is to estimate the joint distribution of $(\mathbf{X}, Y)$. We have to make assumptions about the distribution of $(\mathbf{X}, Y)$, otherwise the problem is ill-defined.

1.5.2 Labeled sequence learning

In the problem of *labeled sequence learning* the phenomenon is a fully observable triplet $(\mathbf{X}, Y, T)$, where

- $\mathbf{X}$ taking values from $\mathbb{R}^d$ is called *input*,
- Y taking values from Y taking values from $\mathcal{C} = \{c_1, \ldots, c_M\}, M \geq 2$ is called *label*, and
- T taking values from $\mathbb{N}$ is called *time*.

It is assumed that the event $T = t$ has nonzero probability for all $t \in \mathbb{N}$. The goal is to predict Y from $(\mathbf{X}, T)$ with a function $g : \mathbb{R}^d \times \mathbb{N} \mapsto \mathcal{C}$ called *classifier* such that the long term error probability

$$L(g) = \sum_{t=0}^{\infty} \mathbf{P}\{g(\mathbf{X}, t) \neq Y | T = t\}$$

is minimal.

1.5.3 Time series prediction

In the problem of *time series prediction* the phenomenon is a fully observable triplet $(\mathbf{X}, Y, T)$, where

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

It is assumed that the event $T = t$ has nonzero probability for all $t \in \mathbb{N}$. The goal is to predict Y from $(\mathbf{X}, T)$ with a function $g : \mathbb{R}^d \times \mathbb{R} \mapsto \mathcal{C}$ called *predictor* such that the long term mean squared error

$$L(g) = \sum_{t=0}^{\infty} \mathbf{E}\{(g(\mathbf{X}, t) - Y)^2 | T = t\}$$

is minimal.

Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth

2 Algorithms

In this chapter I will propose novel algorithms that use convex polyhedrons for solving various machine learning problems. At first, let us consider the problem of convex separability. Assume that $\mathcal{P}$ and $\mathcal{Q}$ are two finite point sets in $\mathbb{R}^d$. One may ask different questions about separating $\mathcal{P}$ and $\mathcal{Q}$:

A) Is there a half-space $\mathcal{S} \subset \mathbb{R}^d$ such that $\mathcal{P} \subset \mathcal{S}$ and $\mathcal{Q} \subset \bar{\mathcal{S}}$?[1]

B) Is there a convex polyhedron $\mathcal{S} \subset \mathbb{R}^d$ such that $\mathcal{P} \subset \mathcal{S}$ and $\mathcal{Q} \subset \bar{\mathcal{S}}$?

C) For fixed K, is there a convex K-polyhedron $\mathcal{S} \subset \mathbb{R}^d$ such that $\mathcal{P} \subset \mathcal{S}$ and $\mathcal{Q} \subset \bar{\mathcal{S}}$?

Question A is known as the problem of *linear separability* and question B as the problem of *convex separability*. We will see that both can be decided in polynomial time. In contrast, answering question C is NP hard[2]. The only easy case is $K = 1$, where the task is to determine linear separability. It is interesting that even the case $K = 2$ called *wedge separation* is NP hard [Megiddo, 1988, Arkin et al., 2006].

Roughly speaking, this implies the following for convex polyhedron classification: if an algorithm using K hyperplanes wants to exactly minimize the number of misclassifications in the training set, then it has to be slow. Only approximate algorithms can be efficient in terms of running time, if the number of hyperplanes is fixed.

I underline that convex separability and convex polyhedron classification are different problems. A convex separability algorithm has only to say a simple yes or no answer. If the algorithm is constructive (shows a polyhedron if the answer is yes), then the only requirement for the constructed polyhedron is to demonstrate convex separability.

In contrast, the output of a convex polyhedron classification algorithm is always a convex polyhedron. The goal is not to separate the classes perfectly in the training set, but to minimize the error probability for future examples. Sometimes it is worthwhile to commit errors in the training set in order to achieve better generalization.

2.1 Linear programming basics

The next section of this chapter will use elements from the theory of linear programming. For those ones who are less familiar with the topic, this section gives an overview about some basic concepts and results (for more details see e.g. [Chvátal, 1983] or [Ferguson, 2004]).

[1] $\bar{\mathcal{S}}$ means $\mathbb{R}^d \setminus \mathcal{S}$, the complement of $\mathcal{S}$.

[2] It is important to note that the dimension d is not fixed.

In a *constrained optimization problem* the task is to maximize or minimize an *objective function* subject to *constraints* on the possible values of the variables. A vector of variables is called *feasible*, if it satisfies the constraints. The set of feasible vectors is called the *feasible set*. A constrained optimization problem is said to be *feasible*, if its feasible set is not empty; otherwise it is termed *infeasible*. A constrained maximization (minimization) problem is said to be *unbounded*, if its objective function can assume arbitrarily large positive (negative) values at feasible vectors; otherwise it is termed *bounded*. Note that infeasible problems are considered as bounded according to the definition.

A *linear program* is a constrained optimization problem, in which the objective function is linear, and the constraints are linear inequalities or equalities. The *standard maximum form* of a linear program is the following:

$$\begin{aligned} \text{variables: } & \mathbf{x} \in \mathbb{R}^d \\ \text{maximize: } & \mathbf{c}^T\mathbf{x} \\ \text{subject to: } & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ contains the variables that should be set, and $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^d$ contain fixed, known values. It is easy to show that every linear program can be transformed to standard maximum form:

- The minimization of function f is equivalent with the maximization of $-f$.
- The constraint $\alpha \geq \beta$ is equivalent with $-\alpha \leq -\beta$.
- The constraint $\alpha = \beta$ is equivalent with $\alpha \leq \beta$, $-\alpha \leq -\beta$.
- An unrestricted variable can be represented as the difference of two non-negative ones.

If maximization is replaced by minimization and $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ is replaced by $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, then we get the *standard minimum form*. A linear program in standard maximum (minimum) form is also called a *standard maximum (minimum) problem*.

The *dual* of the standard maximum problem (2.1) is defined as the following standard minimum problem:

$$\begin{aligned} \text{variables: } & \mathbf{y} \in \mathbb{R}^n \\ \text{minimize: } & \mathbf{b}^T\mathbf{y} \\ \text{subject to: } & \mathbf{A}^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0}, \end{aligned} \tag{2.2}$$

The original standard maximum problem is referred as *primal* in this relation. If we transform (2.2) to standard maximum form (by multiplying $\mathbf{A}$, $\mathbf{c}$, and $\mathbf{c}$ by -1), then its dual by definition is (2.1), transformed to standard minimum form. Therefore, it is rightful to say that (2.1) and (2.2) are the duals of each other.

If a vector $\mathbf{z}$ is a feasible for the primal (dual) problem, then it termed primal (dual) feasible. The *weak duality theorem* says that for any primal feasible $\mathbf{x}$ and dual feasible $\mathbf{y}$

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}.$$

The theorem is a straightforward consequence of the definitions:

1. $\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{A}\mathbf{x}$, because $\mathbf{c} \leq \mathbf{A}^T\mathbf{y}$ and $\mathbf{x} \geq \mathbf{0}$.
2. $\mathbf{y}^T\mathbf{A}\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$, because $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{y} \geq \mathbf{0}$.

It follows directly from the theorem that if the primal and the dual problems are both feasible, then both are bounded too. Moreover, if there exist a primal feasible $\mathbf{x}^*$ and a dual feasible $\mathbf{y}^*$ so that $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ then $\mathbf{x}^*$ is optimal for the primal and $\mathbf{y}^*$ for the dual problem.

The *strong duality theorem* says that if a standard maximum problem is feasible and bounded, then so its dual, and there *exists* a primal feasible $\mathbf{x}^*$ and a dual feasible $\mathbf{y}^*$ so that $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$. The proof (that can be found e.g. in [Chvátal, 1983]) is not as straightforward as the weak duality theorem's.

Now we introduce the concept of equivalence between linear programs in order to obtain a more general definition of duality. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be linear programs with d_1 and d_2 variables. P_1 and P_2 are *equivalent*, if there exists a bijection between $\mathbb{R}^{d_1}$ and $\mathbb{R}^{d_2}$ so that the feasible sets correspond to each other, and

- the objective function values are the same for corresponding vectors, if $\mathcal{P}_1$ and $\mathcal{P}_2$ are both maximization or minimization problems,
- the objective function values equal -1 times each other for corresponding vectors, one of the problems is a maximization and the other is a minimization problem.

For example, let us consider the following linear program:

$$\begin{aligned} \text{variables: } & \mathbf{x} \in \mathbb{R}^d,\ \mathbf{s} \in \mathbb{R}^n \\ \text{maximize: } & \mathbf{c}^T\mathbf{x} \\ \text{subject to: } & \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}. \end{aligned}$$

It is easy to see that this is equivalent with the standard maximum problem (2.1). The bijection is given by the formulae $\mathbf{z} = \mathbf{x}$ and $\mathbf{s} = \mathbf{A}\mathbf{x} - b$. I remark that this formulation is called the *augmented form* of the standard maximum problem, and the variables $s_1, \ldots, s_n$ are called *slack variables*.

Now we are ready introduce the more general definition of duality. Let $\mathcal{P}$ and $\mathcal{Q}$ be two linear programs. $\mathcal{Q}$ is said to be the dual of $\mathcal{P}$, if a standard minimum equivalent of $\mathcal{Q}$ is the dual of a standard maximum equivalent of $\mathcal{P}$.

2.2 Algorithms for determining separability

In this section I will introduce definitions and overview conventional approaches for determining linear and convex separability. I will also propose two new families of methods: one for linear and one for convex separability. In the Applications chapter it will be demonstrated by experiments that the proposed algorithms compare favorably in running time with the existing ones.

2.2.1 Definitions

Let $\mathcal{T}$ be a subset of $\mathbb{R}^d$. The *convex hull* of $\mathcal{T}$ denoted by $\text{conv}(\mathcal{T})$ is the minimal convex subset of $\mathbb{R}^d$ that contains $\mathcal{T}$. If $\mathcal{T}$ is finite, then $\text{conv}(\mathcal{T})$ is a convex polyhedron. A convex polyhedron in $\mathbb{R}^d$ can be given either by its vertices or its $(d-1)$-dimensional facets. The former is called *vertex representation*, and the latter is called *half-space representation*. In high dimensions it is

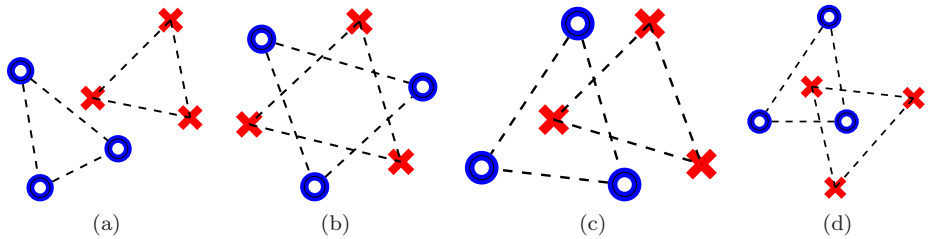


Figure 2.1: Examples for linearly separable (a), mutually convexly separable (b), convexly separable (c), and convexly nonseparable (d) point sets.

typically difficult to change from a given representation to the other, because the convex hull of points can have intractably many facets, and the intersection of half-spaces can assign intractably many extremal points.

Let $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\} \subset \mathbb{R}^d$ and $\mathcal{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\} \subset \mathbb{R}^d$ be two finite point sets.

Definition 2.1. $\mathcal{P}$ and $\mathcal{Q}$ are *linearly separable* if and only if $\text{conv}(\mathcal{P}) \cap \text{conv}(\mathcal{Q}) = \emptyset$.

Definition 2.2. $\mathcal{P}$ and $\mathcal{Q}$ are *convexly separable* if and only if $\mathcal{P} \cap \text{conv}(\mathcal{Q}) = \emptyset$ or $\mathcal{Q} \cap \text{conv}(\mathcal{P}) = \emptyset$. If $\mathcal{P} \cap \text{conv}(\mathcal{Q})$ and $\mathcal{Q} \cap \text{conv}(\mathcal{P})$ are both empty, then $\mathcal{P}$ and $\mathcal{Q}$ are called *mutually convexly separable*. If $\mathcal{Q} \cap \text{conv}(\mathcal{P})$ is empty, but $\mathcal{P} \cap \text{conv}(\mathcal{Q})$ is not, then $\mathcal{P}$ is called the *inner set* and $\mathcal{Q}$ the *outer set*.

Figure 2.1 shows examples for different types of separability. Note that linear separability implies mutual convex separability, but the reverse is not true.

2.2.2 Algorithms for linear separability

The question of linear separability can be formulated as a linear programming problem:

$$\begin{aligned} \text{variables: } & \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \\ \text{minimize: } & 1 \\ \text{subject to: } & \mathbf{w}^T\mathbf{p}_i + b \geq +\varepsilon, \quad i = 1, \ldots, m \\ & \mathbf{w}^T\mathbf{q}_j + b \leq -\varepsilon, \quad j = 1, \ldots, n \end{aligned} \tag{2.3}$$

where ε is an arbitrary positive constant. The constraints express that the elements of $\mathcal{P}$ and $\mathcal{Q}$ have to be on the opposite sides of the hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$. $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the problem is feasible. This basic and straightforward method will be referred as LSEP_1.

Maybe it is a bit unusual in LSEP_1 that the function to minimize is constant. By introducing slack variables we can obtain a formulation that has a non-constant objective function, and that always has a feasible and bounded solution:

$$\begin{aligned} \text{variables: } & \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R},\ \mathbf{s} \in \mathbb{R}^m,\ \mathbf{t} \in \mathbb{R}^n \\ \text{minimize: } & \sum_{i=1}^{m} s_i + \sum_{j=1}^{n} t_j \\ \text{subject to: } & \mathbf{w}^T\mathbf{p}_i + b \geq +\varepsilon - s_i, \quad s_i \geq 0, \quad i = 1, \ldots, m \\ & \mathbf{w}^T\mathbf{q}_j + b \leq -\varepsilon + t_j, \quad t_j \geq 0, \quad j = 1, \ldots, n \end{aligned} \tag{2.4}$$

$\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the solution is $\mathbf{s} = \mathbf{0}, \mathbf{t} = \mathbf{0}$. Linear programming problems can be solved in polynomial time e.g. by using Karmarkar's algorithm [Karmarkar, 1984], therefore linear separability can be decided in polynomial time.

The dual of the LSEP_1 formulation (referred as LSEP_1^*) is the following:

$$\begin{aligned} \text{variables: } & \boldsymbol{\alpha} \in \mathbb{R}^m,\ \boldsymbol{\beta} \in \mathbb{R}^n \\ \text{maximize: } & \varepsilon \left(\sum_{i=1}^{m} \alpha_i + \sum_{j=1}^{n} \beta_j \right) \\ \text{subject to: } & \sum_{i=1}^{m} \alpha_i \mathbf{p}_i = \sum_{j=1}^{n} \beta_j \mathbf{q}_j \\ & \sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j, \quad \boldsymbol{\alpha}, \geq \mathbf{0}, \quad \boldsymbol{\beta} \geq \mathbf{0} \end{aligned} \tag{2.5}$$

Note that the problem is always feasible. If $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are not zero vectors, then the constraints are expressing that the $\text{conv}(\mathcal{P})$ and $\text{conv}(\mathcal{Q})$ have a common point. $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the solution is $\boldsymbol{\alpha} = \mathbf{0}$, $\boldsymbol{\beta} = \mathbf{0}$. If $\mathcal{P}$ and $\mathcal{Q}$ are not linearly separable, then the solution is unbounded.

It is natural to introduce a slightly modified version of LSEP_1^* (referred as LSEP_1^+):

$$\begin{aligned} \text{variables: } & \boldsymbol{\alpha} \in \mathbb{R}^m,\ \boldsymbol{\beta} \in \mathbb{R}^n \\ \text{maximize: } & \varepsilon \left(\sum_{i=1}^{m} \alpha_i + \sum_{j=1}^{n} \beta_j \right) \\ \text{subject to: } & \sum_{i=1}^{m} \alpha_i \mathbf{p}_i = \sum_{j=1}^{n} \beta_j \mathbf{q}_j \\ & \sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j = 1, \quad \boldsymbol{\alpha}, \geq \mathbf{0}, \quad \boldsymbol{\beta} \geq \mathbf{0} \end{aligned} \tag{2.6}$$

The difference from LSEP_1^* is that now the components must sum to 1 in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the problem is infeasible. If $\mathcal{P}$ and $\mathcal{Q}$ are not linearly separable, then the problem has a feasible solution.

An interesting modification of LSEP_1 (referred as LSEP_2) tries to find a separating hyperplane with a small norm:

$$\begin{aligned} \text{variables: } & \mathbf{w}, \mathbf{v} \in \mathbb{R}^d,\ b \in \mathbb{R} \\ \text{minimize: } & \sum_{k=1}^{d} (w_k + v_k) \\ \text{subject to: } & (\mathbf{w} - \mathbf{v})^T \mathbf{p}_i + b \geq +\varepsilon, \quad i = 1, \ldots, m \\ & (\mathbf{w} - \mathbf{v})^T \mathbf{q}_j + b \leq -\varepsilon, \quad j = 1, \ldots, n \\ & \mathbf{w} \geq \mathbf{0}, \quad \mathbf{v} \geq \mathbf{0} \end{aligned} \tag{2.7}$$

$\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the problem is feasible. The price of penalizing the L1 norm of $\mathbf{w}$ and $\mathbf{v}$ is that LSEP_2 has (nearly) twice as many variables as LSEP_1. We will see that this extra computational cost can pay off in certain cases.

The dual of LSEP_2 (referred as LSEP_2^*) is the following:

$$\begin{aligned} &\text{variables: } \boldsymbol{\alpha} \in \mathbb{R}^m,\ \boldsymbol{\beta} \in \mathbb{R}^n \\ &\text{maximize: } \varepsilon \left(\sum_{i=1}^{m} \alpha_i + \sum_{j=1}^{n} \beta_j \right) \\ &\text{subject to: } -\mathbf{1} \leq \sum_{i=1}^{m} \alpha_i \mathbf{p}_i - \sum_{j=1}^{n} \beta_j \mathbf{q}_j \leq \mathbf{1} \\ &\qquad \sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j, \quad \boldsymbol{\alpha}, \geq \mathbf{0}, \quad \boldsymbol{\beta} \geq \mathbf{0} \end{aligned} \tag{2.8}$$

where $\mathbf{1}$ denotes the all-one vector. $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if the solution is $\boldsymbol{\alpha} = \mathbf{0}$, $\boldsymbol{\beta} = \mathbf{0}$.

Finally, a quadratic programming based formulation (referred as LSEP_S) is the following:

$$\begin{aligned} &\text{variables: } \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R},\ \mathbf{s} \in \mathbf{R}^m,\ \mathbf{t} \in \mathbb{R}^n \\ &\text{minimize: } \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \left(\sum_{i=1}^{m} s_i + \sum_{j=1}^{n} t_j \right) \\ &\text{subject to: } \mathbf{w}^T\mathbf{p}_i + b \geq +1 - s_i, \quad s_i \geq 0, \quad i = 1, \ldots, m \\ &\qquad \mathbf{w}^T\mathbf{q}_j + b \leq -1 + t_j, \quad t_j \geq 0, \quad j = 1, \ldots, n \end{aligned} \tag{2.9}$$

Note that this is equivalent with linear SVM training. $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable if and only if there exist a $C > 0$ for which the solution has the following property: $s_1, \ldots, s_m, t_1, \ldots, t_n < 1$. In practice it is not possible to check this property for all C values, just for a reasonably large one. Therefore, we cannot completely rely on the answer, if LSEP_S says no.

Recall that if the point sets are linearly separable (and there are no slack variables), then minimizing $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ means maximizing the distance between the separating hyperplane and the closest points. Obviously, introducing this quadratic term into the objective function makes the optimization problem harder. The rationale behind this formulation is that for linear SVM training there exist efficient solving algorithms and fine-tuned software.

Proposed new methods

The algorithms presented so far try to solve the full problem in one step. Here I propose a novel approach (referred as LSEPX) that is incremental:

1. Let $\mathcal{P}_1, \ldots, \mathcal{P}_L$ and $\mathcal{Q}_1, \ldots, \mathcal{Q}_L$ be two systems of sets such that $\mathcal{P}_1 \subset \cdots \subset \mathcal{P}_L = \mathcal{P}$ and $\mathcal{Q}_1 \subset \cdots \subset \mathcal{Q}_L = \mathcal{Q}$.
2. For $k = 1, \ldots, L$:
   - Check whether $\mathcal{P}_k$ and $\mathcal{Q}_k$ are separable using LSEP_1, or LSEP_2. The result of this step is a yes or no answer and a separating hyperplane $\mathbf{w}_k^T\mathbf{x} + b_k = 0$ if the answer is yes.
   - If the answer is no, then $\mathcal{P}$ and $\mathcal{Q}$ are not linearly separable.
   - If the hyperplane $\mathbf{w}_k^T\mathbf{x} + b_k = 0$ separates $\mathcal{P}$ and $\mathcal{Q}$, then $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable.

$\mathcal{P}_k$ and $\mathcal{Q}_k$ can be called the active sets in the k-th iteration. The last iteration is equivalent with solving the full problem. The advantage of the approach is that there is a chance of getting the answer before the last iteration. However, there is no guarantee for that. A reasonable heuristic for defining the active sets is the following:

1. $\mathcal{P}_1$, $\mathcal{Q}_1 \leftarrow$ random $\min\{d, |\mathcal{P}|\}$ and $\min\{d, |\mathcal{Q}|\}$ element subsets of $\mathcal{P}$ and $\mathcal{Q}$.
2. At the k-th iteration:
   - For each $\mathbf{x} \in \mathcal{P} \cup \mathcal{Q}$ calculate $\delta_k(\mathbf{x}) = (\mathbf{w}_k^T \mathbf{x} + b)(-1)^{I\{\mathbf{x} \in \mathcal{Q}\}}$.
   - Denote the set of γ_k points with smallest δ_k values by $\mathcal{U}_k$.
   - $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cup (\mathcal{U}_k \cap \mathcal{P})$, $\quad \mathcal{Q}_{k+1} \leftarrow \mathcal{Q}_k \cup (\mathcal{U}_k \cap \mathcal{Q})$.

Thus, in each iteration the points with largest "errors" are added to the active sets. Some possible choices for γ_k are $\gamma_k \equiv 1$, $\gamma_k \equiv d$, or $\gamma_k = 2^k d$.

A possible disadvantage of LSEPX is that points are never removed from the active sets. As a consequence, the active sets may contain redundant elements, which can increase running time. On the other hand, if we allow removals from the active sets without restrictions, then there is no guarantee for stopping. A simple solution to the dilemma is to allow removing points only once.

The modified version of LSEPX (referred as LSEPY) defines the active sets as follows:

1. $\mathcal{P}_1$, $\mathcal{Q}_1 \leftarrow$ random $\min\{d, |\mathcal{P}|\}$ and $\min\{d, |\mathcal{Q}|\}$ element subsets of $\mathcal{P}$ and $\mathcal{Q}$.
2. At the k-th iteration:
   - For each $\mathbf{x} \in \mathcal{P} \cup \mathcal{Q}$ calculate $\delta_k(\mathbf{x}) = (\mathbf{w}_k^T \mathbf{x} + b)(-1)^{I\{\mathbf{x} \in \mathcal{Q}\}}$.
   - Denote the set of γ_k points with smallest δ_k values by $\mathcal{U}_k$.
   - Denote the set of points with δ_k value greater than $\varepsilon_2 > 0$ by $\mathcal{V}_k$.
   - $\mathcal{V}_k \leftarrow \mathcal{V}_k \setminus \left(\cup_{l=1}^{k-1} \mathcal{V}_l\right)$, in order to avoid multiple removals.
   - Denote the element of $\mathcal{P}$ and $\mathcal{Q}$ with minimal δ_k value by $\mathbf{p}'$ and $\mathbf{q}'$. Remove $\mathbf{p}'$ and $\mathbf{q}'$ from $\mathcal{V}_k$, in order to keep at least 1 point from $\mathcal{P}$ and $\mathcal{Q}$.
   - $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cup (\mathcal{U}_k \cap \mathcal{P}) \setminus (\mathcal{V}_k \cap \mathcal{P})$, $\quad \mathcal{Q}_{k+1} \leftarrow \mathcal{Q}_k \cup (\mathcal{U}_k \cap \mathcal{Q}) \setminus (\mathcal{V}_k \cap \mathcal{Q})$.

It is also possible to introduce an incremental method based on the dual based formulation LSEP_1^+. The outline of the algorithm (referred as LSEPZ) is the following:

1. Create reduced versions of $\mathcal{P}$ and $\mathcal{Q}$ by keeping only γ randomly selected coordinates (features). Denote the result by $\mathcal{P}^1$ and $\mathcal{Q}^1$.
2. For $k = 1, \ldots n$:
   - Check whether $\mathcal{P}^k$ and $\mathcal{Q}^k$ are separable using LSEP_1^+. The result of this step is a yes or no answer and an $\boldsymbol{\alpha}$ and a $\boldsymbol{\beta}$ vector, if the answer is no.
   - If the answer is yes, then $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable.
   - If the answer is no, then:
     - If $\mathcal{P}_k = \mathcal{P}$ and $\mathcal{Q}_k = \mathcal{Q}$, then $\mathcal{P}$ and $\mathcal{Q}$ are not linearly separable.
     - Calculate $\mathbf{s} = \sum_{i=1}^m \alpha_i \mathbf{p}_i - \sum_{j=1}^n \beta_j \mathbf{q}_j$, where $\mathbf{p}_i$-s and $\mathbf{q}_j$-s are from the original $\mathcal{P}$ and $\mathcal{Q}$ sets.
     - Denote the coordinates with largest $|s_k|$ values by $\mathcal{U}^k$.

* Define $\mathcal{P}^{k+1}$ and $\mathcal{Q}^{k+1}$ as the extension of $\mathcal{P}^k$ and $\mathcal{Q}^k$ with the coordinates in $\mathcal{U}^k$.

It is interesting to observe that the dual based LSEPZ is not perfectly "symmetric" to the previous two primal based approaches. While LSEPX and LSEPY are able to achieve a speedup both in the separable and the nonseparable case, LSEPZ is capable of that only in the nonseparable case. Note that only LSEP_1^+ can be used among the dual based basic methods in LSEPZ, since LSEP_1^* and LSEP_2^* can have unbounded solution in the nonseparable case.

Finally, I would like to mention that it is possible to define hybrid methods (referred as LSEPZX and LSEPZY) based on the previous algorithms:

1. Run LSEPZ and try to reduce the number of coordinates (features).
2. If the answer of LSEPZ is yes, then run LSEPX or LSEPY on the reduced dataset.

2.2.3 Algorithms for convex separability

Recall that $\mathcal{P}$ and $\mathcal{Q}$ are called convexly separable, if and only if $\mathcal{P} \cap \text{conv}(\mathcal{Q}) = \emptyset$ or $\mathcal{Q} \cap \text{conv}(\mathcal{P}) = \emptyset$. Without loss of generality assume that we want to decide whether $\mathcal{Q} \cap \text{conv}(\mathcal{P})$ is empty. The other property can be checked exactly the same way.

At first we overview a naive approach with exponential time complexity:

1. Compute a half-space representation of $\text{conv}(\mathcal{P})$. This yields $\mathbf{w}_k$-s and b_k-s ($k = 1, \ldots, r$) such that $\text{conv}(\mathcal{P}) = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}_1^T\mathbf{x} \geq b_1, \ldots, \mathbf{w}_r^T\mathbf{x} \geq b_r\}$.
2. $\mathcal{Q} \cap \text{conv}(\mathcal{P}) = \emptyset$ if and only if $\max_{j=1,\ldots,n}\{\min_{k=1,\ldots,r}\{\mathbf{w}_k^T\mathbf{q}_j - b_k\}\} < 0$.

The problem with this algorithm is that the size of half-space representation r typically grows exponentially with d.

The methods presented from now will all have polynomial time complexity. A basic and straightforward approach (referred as CSEP) is to separate each element of $\mathcal{Q}$ from $\mathcal{P}$ individually. The primal based version of the algorithm is the following:

1. $\mathcal{U} \leftarrow \emptyset$, $\quad \mathbf{q} \leftarrow$ a random element of $\mathcal{Q}$.
2. For $k = 1, \ldots, n$:
   - Check whether $\mathcal{P}$ and $\{\mathbf{q}\}$ are linearly separable using LSEP_1, or LSEP_2, LSEPX, LSEPY, LSEPZX or LSEPZY. The result of this step is a yes or no answer and a separating hyperplane $\mathbf{w}_k^T\mathbf{x} + b_k = 0$ if the answer is yes.
   - If the answer is no, then $\mathcal{P}$ and $\mathcal{Q}$ are not convexly separable.
   - If the answer is yes, then:
     * If $\mathcal{U} = \mathcal{Q}$, then $\mathcal{P}$ and $\mathcal{Q}$ are convexly separable
     * For each $\mathbf{x} \in \mathcal{Q} \setminus \mathcal{U}$ calculate $\delta_k(\mathbf{x}) = -(\mathbf{w}_k^T\mathbf{x} + b_k)$.
     * If the smallest δ_k value is greater than 0, then $\mathcal{P}$ and $\mathcal{Q}$ are convexly separable.
     * Add the point with smallest δ_k value to $\mathcal{U}$.

The dual based version of CSEP is the following:

1. For $k = 1, \ldots, n$:
   - Check whether $\mathcal{P}$ and $\{\mathbf{q}_k\}$ are linearly separable using LSEP_1^*, LSEP_1^+, LSEP_2^*, or LSEPZ. The result of this step is a yes or no answer.

– If the answer is no, then $\mathcal{P}$ and $\mathcal{Q}$ are not convexly separable.

2. $\mathcal{P}$ and $\mathcal{Q}$ are convexly separable.

Note that the primal based version is able to finish in less than n iterations both in the separable and the nonseparable case. In contrast, the dual based version always runs $|\mathcal{Q}|$ iterations, if $\mathcal{P}$ and $\mathcal{Q}$ are convexly separable.

A proposed new method

At first I introduce a fast algorithm (referred as CSEPC) that performs approximate convex separation:

1. $\mathcal{V} \leftarrow \emptyset$. $s_1, \ldots, s_n \leftarrow \infty$.

2. Compute the centroid of $\mathcal{P}$ as $\bar{\mathbf{p}} \leftarrow \frac{1}{m}(\mathbf{p}_1 + \cdots + \mathbf{p}_m)$.

3. Choose $\mathbf{q}_k$ from $\mathcal{Q}$ such that $k = \arg\max_{j=1,\ldots,n}\{s_j\}$.

4. If $s_k \leq 0$, then return $\mathcal{V}$.

5. Compute $\mathbf{w} \leftarrow (\bar{\mathbf{p}} - \mathbf{q}_k)/\|\bar{\mathbf{p}} - \mathbf{q}_k\|$ and $b \leftarrow \min_{i=1,\ldots,m}\{\mathbf{w}^T\mathbf{p}_i\}$.

6. $\mathcal{V} \leftarrow \mathcal{V} \cup \{(\mathbf{w}, b)\}$.

7. For all $s_j > 0$: $s_j \leftarrow \min\{s_j,\ \mathbf{w}^T\mathbf{q}_j - b\}$. $s_k \leftarrow 0$.

8. Go to step 4.

The idea of the algorithm is to define hyperplanes by connecting the elements of $\mathcal{Q}$ with the centroid of $\mathcal{P}$, and translate the hyperplanes to the boundary of $\mathrm{conv}(\mathcal{P})$. Therefore the algorithm can be called the *centroid method*. The centroid method is often able to separate most of the elements of $\mathcal{Q}$ from $\mathrm{conv}(\mathcal{P})$. However, it does not guarantee to find a convex separation even for for convexly separable point sets.

Now I propose an exact algorithm (referred as CSEPX) that uses the centroid method as a preprocessor:

1. Run CSEPC on $\mathcal{P}, \mathcal{Q}$. The result of this step is a set of weight–bias pairs $\mathcal{V} = \{(\mathbf{w}_1, b_1), \ldots, (\mathbf{w}_r, b_r)\}$ and a set of not separated points $\mathcal{Q}' = \{\mathbf{q} \in \mathcal{Q} : \min_{k=1,\ldots,r}\{\mathbf{w}_k^T\mathbf{q} + b_k\} \geq 0\}$.

2. Run CSEP on $\mathcal{P}, \mathcal{Q}'$. The result of this step is a yes or no answer A indicating whether $\mathcal{Q}' \cap \mathrm{conv}(\mathcal{P})$ is empty, and a set of weight–bias pairs $\mathcal{W}$, if the answer is yes.

3. If A is no, then answer no. If A is yes, then answer yes and return $\mathcal{V} \cup \mathcal{W}$.

CSEPC and CSEPX were first published in [Takács and Pataki, 2007a]. In many practical cases CSEPX can achieve a large speedup over the other presented methods. The efficiency of the algorithm will demonstrated by experiments in the Applications chapter.

2.3 Algorithms for classification

Recall, that a convex K-polyhedron classifier is function $g : \mathbb{R}^d \mapsto \{c_1, c_2\}$ that can be written in the following form:

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

where $\mathbf{w}_1, \ldots, \mathbf{w}_K \in \mathbb{R}^d$ are called weight vectors and $b_1, \ldots, b_K \in \mathbb{R}$ are called biases. The class associated with c_1 is called the positive, and the class associated with c_2 the negative class. It is assumed that the labels are $c_1 = 1$, $c_2 = 0$, and also that the negative class has higher probability ($\mathbf{P}\{Y = 0\} > \mathbf{P}\{Y = 1\}$).

If $\mathbf{w}_k^T\mathbf{x} < -b_k$ for any $k \in \{1, \ldots, K\}$, then the input $\mathbf{x}$ can be classified as negative immediately. Therefore, convex polyhedron classifiers tend to classify negative examples quickly, which makes the approach particularly suitable for unbalanced problems.

Despite this appealing property, convex polyhedron classifiers are not frequently used in practice currently. 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.

2.3.1 Known methods

Probably the best known work that applied convex polyhedron classifiers for solving a practical problem is [Elad et al., 2001]. 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

$$\mathcal{M}(\mathbf{w}) = \frac{(\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_0)^2 + \mathbf{w}^T\mathbf{R}_1\mathbf{w} + \mathbf{w}^T\mathbf{R}_0\mathbf{w}}{\mathbf{w}^T\mathbf{R}_1\mathbf{w} + \lambda\mathbf{w}^T\mathbf{w}}, \tag{2.10}$$

where $\mathbf{m}_1$, $\mathbf{m}_0$, $\mathbf{R}_1$ and $\mathbf{R}_0$ are the empirical means and covariances of the classes (see the description of Fisher discriminant analysis on page 14 for the details), and λ is the regularization coefficient. 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[3]:

- For $k = 1, \ldots, K$:
  - 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 $\sum_{i:y_i=1} I\{g_k(\mathbf{x}_i) \neq y_i\} + \beta \sum_{i:y_i=0} I\{g_k(\mathbf{x}_i) \neq y_i\}$.

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

– 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:

- [Pilászy and Dobrowiecki, 2007] tries to separate each negative example from the positive class individually with an adaptation of the multiclass SVM method [Crammer and Singer, 2001]. The algorithm can be used only for small problems due to its large computational complexity.
- The ID3 algorithm (see page 17) 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.
- The hinging hyperplanes approach [Breiman, 1993] describes one of the classes as the union of convex 2-polyhedrons. A hinge function is the maximum (or minimum) of two linear functions and the hinge is the intersection of the two hyperplanes defined by the linear functions. The classification formula consists of a threshold function applied on the sum of hinge functions. Training is done by iterating over the hinge functions, and setting their parameters separately. The disadvantage of the approach is that it does not deal with K-polyhedrons, and it allows only axis-parallel hinge directions.
- In the literature of *probably approximately correct learning* (PAC learning) [Valiant, 1984] one can find theoretical works related to convex polyhedron classification, for example [Fischer, 1995, Vempala, 1997, Kwek and Pitt, 1998, Klivans et al., 2004]. 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.

2.3.2 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[4] 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:

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

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:

$$\frac{f(u_j)}{f(u_{\max})} = \frac{f(u_j)}{f(u_j + (u_{\max} - u_j))} \approx \begin{cases} 0 & \text{if } u_j \neq u_{\max}, \\ 1 & \text{if } u_j = u_{\max}. \end{cases} \quad (2.11)$$

[4] Infinitely many times differentiable.

It follows from (2.11) that

$$\frac{f(u_j)}{\sum_{k=1}^{K} f(u_k)} = \frac{f(u_j)/f(u_{\max})}{\sum_{k=1}^{K} f(u_k)/f(u_{\max})} \approx \begin{cases} 0 & \text{if } u_j \neq u_{\max}, \\ 1 & \text{if } u_j = u_{\max}. \end{cases} \tag{2.12}$$

If f is monotonically increasing and smooth, then based on (2.12) it is possible to define smooth approximations for the maximum function:

$$\begin{aligned} &\text{A)}\ \max\{u_1,\dots,u_K\} \approx f^{-1}\left(\sum_{k=1}^{K} f(u_k)\right), \\ &\text{B)}\ \max\{u_1,\dots,u_K\} \approx f^{-1}\left(\frac{1}{K}\sum_{k=1}^{K} f(u_k)\right), \\ &\text{C)}\ \max\{u_1,\dots,u_K\} \approx \sum_{j=1}^{K} \frac{f(u_j)}{\sum_{k=1}^{K} f(u_k)} u_j. \end{aligned} \tag{2.13}$$

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:

$$\begin{aligned} \text{smax}_{A1}(\mathbf{u}) &= \frac{1}{\alpha}\ln\left(\sum_{k=1}^{K}\exp(\alpha u_k)\right) \\ \text{smax}_{A2}(\mathbf{u}) &= \left(\sum_{k=1}^{K} u_k^{\alpha}\right)^{1/\alpha} \\ \text{smax}_{B1}(\mathbf{u}) &= \frac{1}{\alpha}\ln\left(\frac{1}{K}\sum_{k=1}^{K}\exp(\alpha u_k)\right) \\ \text{smax}_{B2}(\mathbf{u}) &= \left(\frac{1}{K}\sum_{k=1}^{K} u_k^{\alpha}\right)^{1/\alpha} \\ \text{smax}_{C1}(\mathbf{u}) &= \sum_{j=1}^{K}\frac{\exp(\alpha u_j)}{\sum_{k=1}^{K}\exp(\alpha u_k)} u_j \\ \text{smax}_{C2}(\mathbf{u}) &= \sum_{j=1}^{K}\frac{u_j^{\alpha}}{\sum_{k=1}^{K} u_k^{\alpha}} u_j \end{aligned} \tag{2.14}$$

where $\mathbf{u} = [u_1,\dots,u_K]$ 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},

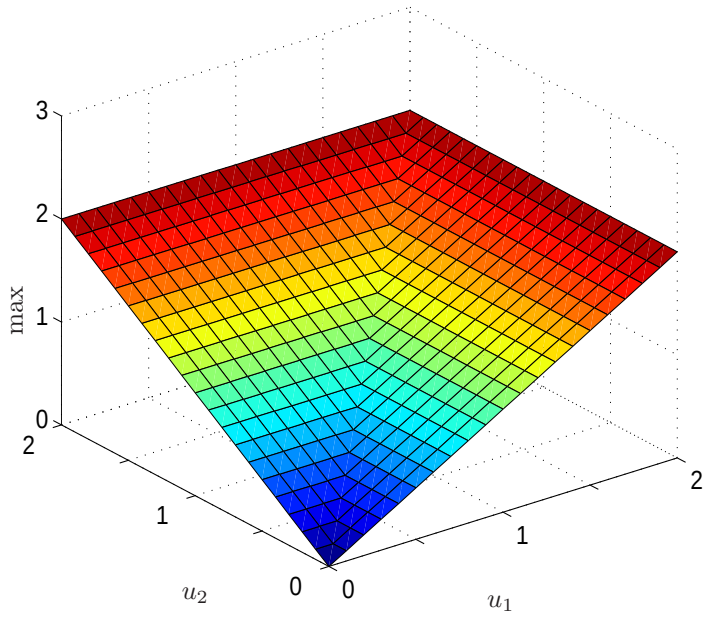


Figure 2.2: The maximum function in 2 dimensions.

smax_{C2} are admissible only if $u_1, \ldots, u_K$ are all non-negative [5]. The surface plot of the maximum function in 2 dimensions can be seen in Figure (2.2). The presented smooth maximum functions are depicted in Figure (2.3) and their difference from max in Figure (2.4).

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

$$\max\{u_1 + C, \ldots, u_K + C\} = \max\{u_1, \ldots, u_K\} + C,$$
$$\max\{Cu_K, \ldots, Cu_K\} = C \max\{u_1, \ldots, u_K\},$$

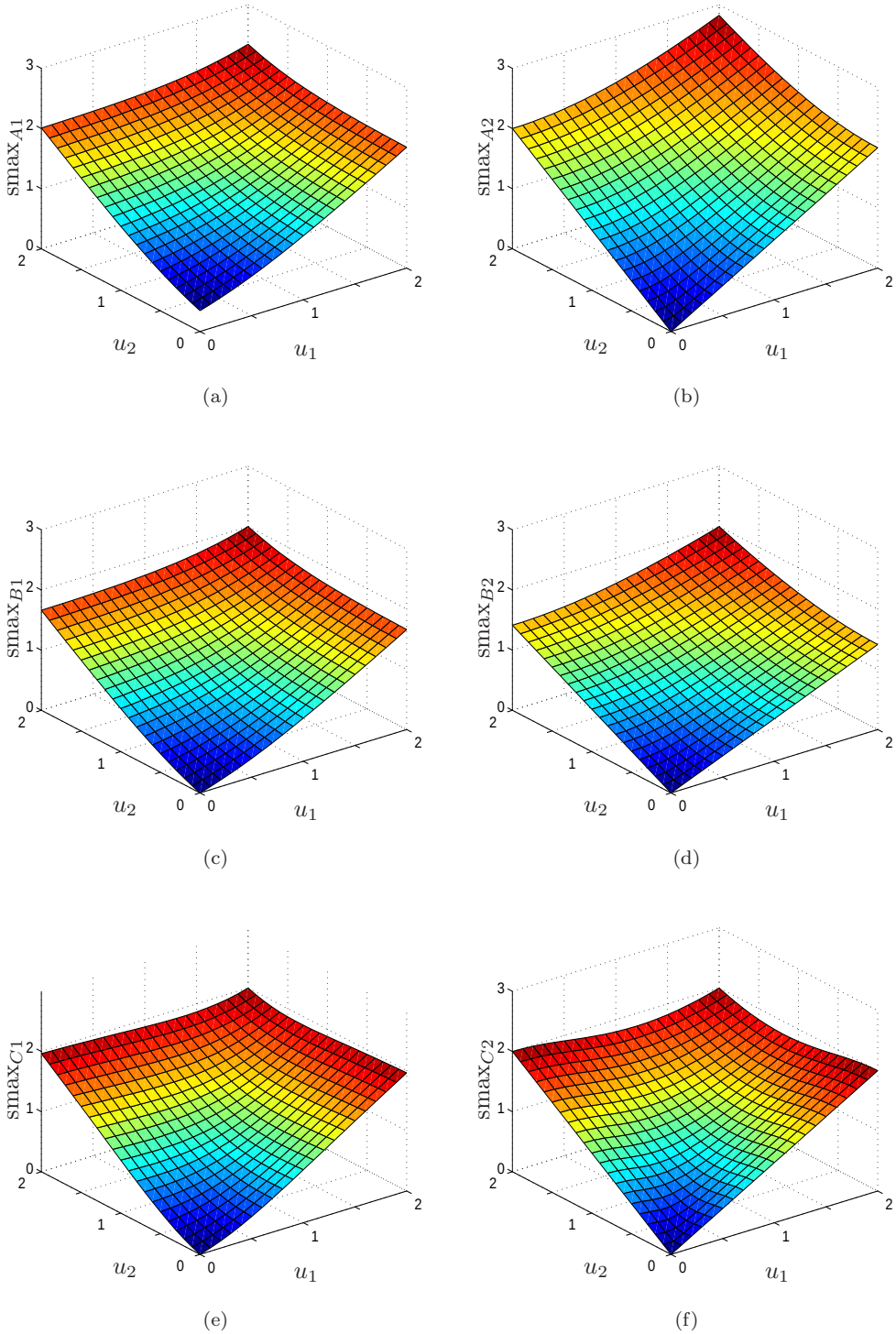
where C is an arbitrary constant in the first case and a non-negative constant in the second case. Interestingly, for 5 of the given smooth maximum functions exactly *one* of these properties is true (smax_{A1}, smax_{B1} and smax_{C1} have the first, smax_{B2} and smax_{C2} have the second property).

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 $p_j = \frac{u_j^\alpha}{\sum_{k=1}^{K} u_k^\alpha}$.

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



(a) (b) (c) (d) (e) (f)

Figure 2.3: Smooth maximum functions in 2 dimensions ($\alpha = 2$).

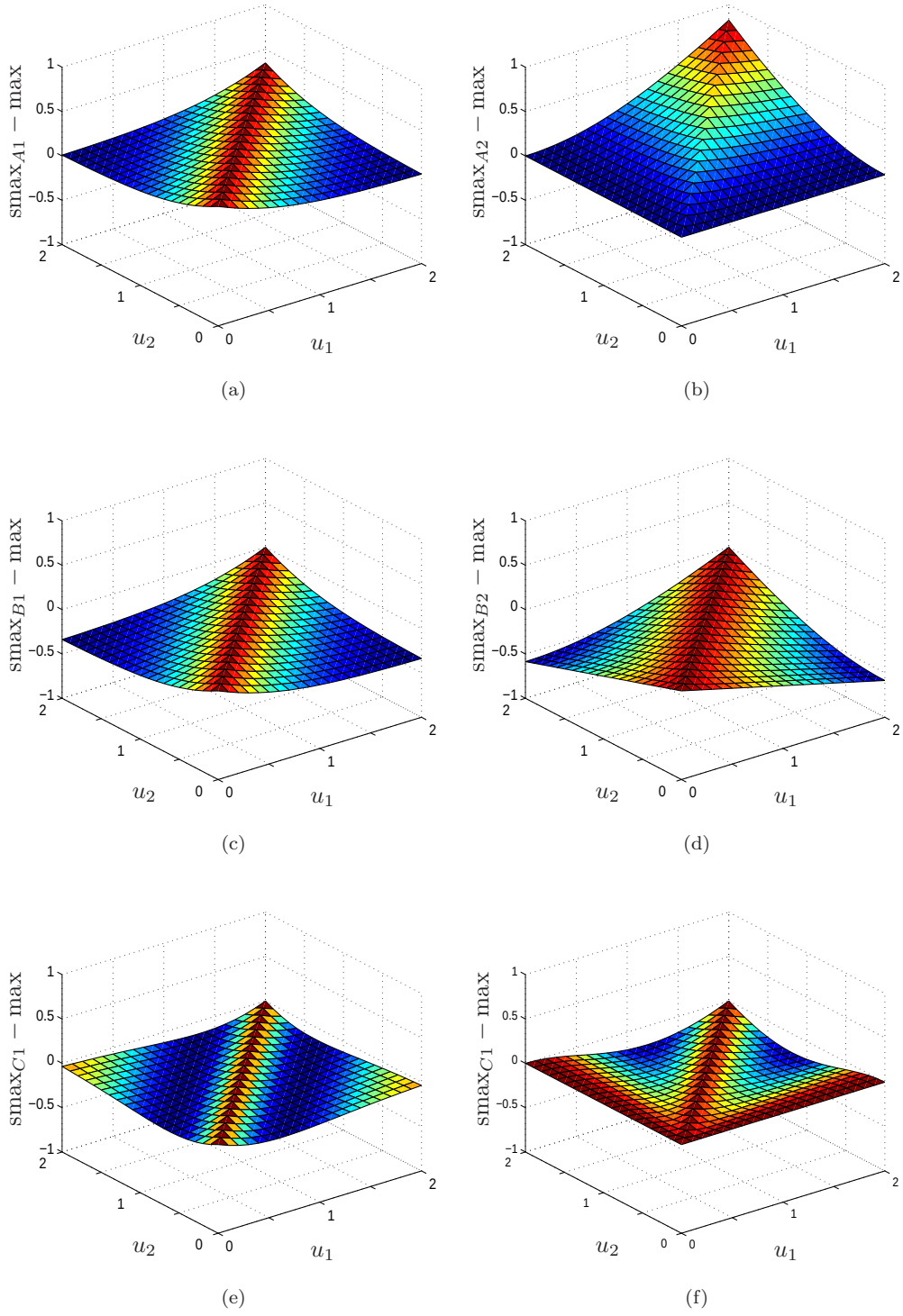


Figure 2.4: The error of smooth maximum functions in 2 dimensions ($\alpha = 2$).

The partial derivatives of the proposed smooth maximum functions are ($j = 1, \ldots, K$):

$$\begin{aligned}
\mathrm{smax}'_{j,A1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{A1}}{\partial u_j}(\mathbf{u}) = p_j, \\
\mathrm{smax}'_{j,A2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{A2}}{\partial u_j}(\mathbf{u}) = p_j \frac{s}{u_j}, \\
\mathrm{smax}'_{j,B1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{B1}}{\partial u_j}(\mathbf{u}) = p_j, \\
\mathrm{smax}'_{j,B2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{B2}}{\partial u_j}(\mathbf{u}) = p_j \frac{s}{K u_j}, \\
\mathrm{smax}'_{j,C1}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{C1}}{\partial u_j}(\mathbf{u}) = p_j \left(1 + \alpha(u_j - s)\right), \\
\mathrm{smax}'_{j,C2}(\mathbf{u}) &= \frac{\partial \mathrm{smax}_{C2}}{\partial u_j}(\mathbf{u}) = p_j \left(1 + \alpha \left(1 - \frac{s}{u_j}\right)\right),
\end{aligned} \tag{2.15}$$

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).

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 ($j, k = 1, \ldots, K$):

$$\begin{aligned}
\mathrm{smax}''_{jk,A1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{A1}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\alpha, \\
\mathrm{smax}''_{jk,A2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{A2}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\frac{(\alpha - 1)s}{u_j u_k}, \\
\mathrm{smax}''_{jk,B1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{B1}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\alpha, \\
\mathrm{smax}''_{jk,B2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{B2}}{\partial u_j \partial u_k}(\mathbf{u}) = (-p_j p_k + \delta_{jk} p_j)\frac{(\alpha - 1)s}{K^2 u_j u_k}, \\
\mathrm{smax}''_{jk,C1}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{C1}}{\partial u_j \partial u_k}(\mathbf{u}) = \left(-p_j s'_k - p_k s'_j + \delta_{jk}(s'_j + p_j)\right)\alpha, \\
\mathrm{smax}''_{jk,C2}(\mathbf{u}) &= \frac{\partial^2 \mathrm{smax}_{C2}}{\partial u_j \partial u_k}(\mathbf{u}) = \left(-\frac{p_j s'_k}{u_j} - \frac{p_k s'_j}{u_k} + \delta_{jk}\left(\frac{s'_j}{u_j} + \frac{p_j s}{u_j^2}\right)\right)\alpha,
\end{aligned} \tag{2.16}$$

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).

2.3.3 Smooth maximum based algorithms

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, since max can be replaced with a smooth max and $\mathrm{th}(\gamma)$ with $\mathrm{sgm}(\gamma)$ or $\gamma + 0.5$. 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:

$$\begin{aligned} h_{A1}(z) &= -z, & h_{A2}(s) &= \mathrm{sgm}(-s), \\ h_{B1}(z) &= -z, & h_{B2}(s) &= -s + 0.5, \\ h_{C1}(z) &= 1 - \mathrm{sgm}(z), & h_{C2}(s) &= -s + 1. \end{aligned} \tag{2.17}$$

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

$$\begin{aligned} h'_{A1}(z) &= -1, & h'_{A2}(s) &= -h_{A2}(s)(1 - h_{A2}(s)), \\ h'_{B1}(z) &= -1, & h'_{B2}(s) &= -1, \\ h'_{C1}(z) &= -h_{C1}(z)(1 - h_{C1}(z)), & h'_{C2}(s) &= -1, \end{aligned} \tag{2.18}$$

$$\begin{aligned} h''_{A1}(z) &= 0, & h''_{A2}(s) &= -h'_{A2}(s)(1 - 2h_{A2}(s)), \\ h''_{B1}(z) &= 0, & h''_{B2}(s) &= 0, \\ h''_{C1}(z) &= h'_{C1}(z)(1 - 2h_{C1}(z)), & h''_{C2}(s) &= 0. \end{aligned} \tag{2.19}$$

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:

$$\begin{aligned} \mathrm{loss}_S(a, y) &= \tfrac{1}{2}(a - y)^2, \\ \mathrm{loss}_L(a, y) &= -\ln\left(a^y(1 - a)^{1-y}\right). \end{aligned} \tag{2.20}$$

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:

$$\begin{aligned} \text{loss}'_S(a, y) &= \frac{\partial \text{loss}_S}{\partial a}(a, y) = a - y, \\ \text{loss}'_L(a, y) &= \frac{\partial \text{loss}_L}{\partial a}(a, y) = \frac{1 - y}{1 - a} - \frac{y}{a}, \end{aligned} \tag{2.21}$$

$$\begin{aligned} \text{loss}''_S(a, y) &= \frac{\partial^2 \text{loss}_S}{\partial^2 a^2}(a, y) = 1, \\ \text{loss}''_S(a, y) &= \frac{\partial^2 \text{loss}_L}{\partial^2 a^2}(a, y) = \frac{1 - y}{(1 - a)^2} - \frac{y}{a^2}. \end{aligned} \tag{2.22}$$

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

$$\mathcal{L}(b_1, \mathbf{w}_1, \ldots, b_K, \mathbf{w}_K) = \left(\sum_{i=1}^{n} \text{loss}(h(\mathbf{x}_i), y_i) \right) + \lambda \left(\frac{1}{2} \sum_{j=1}^{K} \mathbf{w}_j^T \mathbf{w}_j \right), \tag{2.23}$$

where $\text{loss} \in \{\text{loss}_S, \text{loss}_L\}$, and λ is called regularization coefficient. The number of allowed choices for $(\text{smax}, h, \text{loss})$ 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 SMAX in the rest of the thesis.

It is worthwhile to mention that SMAX contains various linear classification methods as special cases:

- If $K = 1$, $h \in \{h_A, h_C\}$, and $\text{loss} = \text{loss}_L$, then SMAX is equivalent with LOGR.
- If $K = 1$, $h \in \{h_A, h_C\}$, and $\text{loss} = \text{loss}_S$, then SMAX is equivalent with SPER.
- If $K = 1$, $h = h_B$, and $\text{loss} = \text{loss}_S$, then SMAX is equivalent with ALN.

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 (2.5).

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

- $\text{smax} \in \{\text{smax}_{A1}, \text{smax}_{A2}, \text{smax}_{B1}, \text{smax}_{B2}, \text{smax}_{C1}, \text{smax}_{C2}\}$: 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,
- $\text{loss} \in \{\text{loss}_S, \text{loss}_L\}$: per example loss function,
- $K \in \mathbb{N}$: number of hyperplanes in the convex polyhedron classifier,

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 2.5: Stochastic gradient descent with momentum for training 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 (2.6). 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. In the presented version of the algorithm the S step sizes does not depend on each other. Of course it would be possible to replace this simple solution to a more sophisticated one like golden section search [Kiefer, 1953].

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.

Handling missing data

The previous algorithms assume that the phenomenon is fully observable. However, there are many real-world problems (e.g. in the medical domain) in which this assumption is not true, and the training examples contain unknown feature values.

Finding a specialized technique for handling this difficulty that fits well to convex polyhedron classifiers is out of the scope of this thesis. If the training set is incomplete, then I recommend to use a well-known, simple heuristic for handling missing data. Some possible choices are:

- Replacing the missing values with zero.
- Replacing the missing values with the empirical mean or median of the given feature.
- Using one of the previous methods and introducing new binary features that indicate if the value of the original feature was known.

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 $j \leftarrow 1$ **to** K **do**
        for $k \leftarrow 1$ **to** K **do**
            $c''_{jk} \leftarrow \text{loss}''(a, y_i) \cdot h'_2(s)^2 \cdot \text{smax}'_j(\mathbf{u})\text{smax}'_k(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
                $\text{loss}'(a, y_i) \cdot h''_2(s) \cdot \text{smax}'_j(\mathbf{u})\text{smax}'_k(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
                $\text{loss}'(a, y_i) \cdot h'_2(s) \cdot \text{smax}''_{jk}(\mathbf{u}) \cdot h'_1(z_j)h'_1(z_k) +$
                $\text{loss}'(a, y_i) \cdot h'_2(s) \cdot \text{smax}'_j(\mathbf{u})\delta_{jk} \cdot h''_1(z_j)\delta_{jk}$

            for $l \leftarrow 0$ **to** d **do** // update Hessian
                for $m \leftarrow 0$ **to** d **do**
                    $\widehat{j} \leftarrow (j-1)(d+1) + l + 1$
                    $\widehat{k} \leftarrow (k-1)(d+1) + m + 1$
                    $h_{\widehat{j}\widehat{k}} \leftarrow h_{\widehat{j}\widehat{k}} + c''_{jk}x_{il}x_{im} + \lambda\delta_{l0}\delta_{m0}$
                end
            end
        end
    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
    $\mathbf{v} \leftarrow [(b_1 w_{11} \cdots w_{1d}) \cdots (b_K w_{K1} \cdots w_{Kd})]^T$
    $\mathbf{g} \leftarrow [(b'_1 w'_{11} \cdots w'_{1d}) \cdots (b'_K w'_{K1} \cdots w'_{Kd})]^T$
    for σ **in** $\{1, 2^{-1}, \ldots, 2^{-S+2}, 0\}$ **do** // try S step sizes
        $\mathbf{v}_{\text{new}} \leftarrow \mathbf{v} - \sigma\mathbf{H}^{-1}\mathbf{g}$
        $\mathcal{L}_{\text{new}} \leftarrow \mathcal{L}(\mathbf{v}_{\text{new}})$ // use (2.23)
        if $\mathcal{L}_{\text{new}} < \mathcal{L}_{\min}$ **then** $\mathcal{L}_{\min} \leftarrow \mathcal{L}_{\text{new}}$, $\mathbf{v}_{\text{best}} \leftarrow \mathbf{v}_{\text{new}}$
    end
    $[(b_1 w_{11} \cdots w_{1d}) \cdots (b_K w_{K1} \cdots w_{Kd})] \leftarrow \mathbf{v}_{\text{best}}^T$ // 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 2.6: Newton's method for training the convex polyhedron classifier.

2.4 Algorithms for regression

Recall that a convex K-polyhedron predictor is a function $g : \mathbb{R}^d \mapsto \mathbb{R}$ that can be written in the following form:

$$g(\mathbf{x}) = -\max\{-\mathbf{w}_1^T\mathbf{x} - b_1, \ldots, -\mathbf{w}_K^T\mathbf{x} - b_K\}.$$

Analogously with classification, we can 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.

2.5 Algorithms for collaborative filtering

Recall that in the case of collaborative filtering 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

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

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

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 SMAX_{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

$$\begin{aligned}
\frac{\partial \mathcal{L}_{ui}}{\partial p_{ul}^{(k)}}(\mathbf{w}) &= (h(u,i) - r_{ui})(\text{smax}'_k(\mathbf{z})q_{ui}) + \lambda_U p_{ul}^{(k)}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial q_{li}}(\mathbf{w}) &= (h(u,i) - r_{ui}) \left(\sum_{k=1}^{K} \text{smax}'_k(\mathbf{z}) p_{ul}^{(k)} \right) + \lambda_I q_{li}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial b_u}(\mathbf{w}) &= h(u,i) - r_{ui}, \\
\frac{\partial \mathcal{L}_{ui}}{\partial c_i}(\mathbf{w}) &= h(u,i) - r_{ui},
\end{aligned} \tag{2.24}$$

Note that the second equation builds upon the assumption $\text{smax} \in \{\text{smax}_{A1}, \text{smax}_{B1}, \text{smax}_{C1}\}$, 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 (2.7). 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.

Relationship with Mangasarian's results

If we are talking about optimization techniques applied to solve machine learning problems, then we should definitely mention the work of Mangasarian and his colleagues. They have been working in this field since long ago, and they have published many interesting results. An incomplete survey about their results can be found e.g. in [Bradley et al., 1999].

Although there are some intersections between that article and this thesis, generally the two works investigate different problems. The main differences are the following:

- This thesis contains numerous algorithms for determining the linear and convex separability of point sets. That paper contains some basic algorithms for linear separability, but instead of separability they mainly focus on classification, regression, clustering, and dependency modeling.
- They do not investigate the problem of convex separability.
- They usually formulate learning algorithms as constrained optimization problems. In this thesis the proposed classification and regression algorithms are all formulated as unconstrained optimization problems.
- They propose several interesting algorithms for training linear models but they do not deal with training convex polyhedron models.

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** $s'_k \leftarrow \text{smax}'_k(-\mathbf{z})$
      $a \leftarrow b_u + c_i - \text{smax}(-\mathbf{z})$ // 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 2.7: Stochastic gradient descent for training the convex polyhedron predictor.

Technical skill is a mastery of complexity, while creativity is a mastery of simplicity.

Erik Christopher Zeeman

3 Model complexity

From a point of view, machine learning can be seen as modeling. The input is a dataset that was collected by observing a phenomenon. The output is a model that explains certain aspects of the phenomenon, and that can be used for making prediction.

In a typical machine learning project many experiments are performed and many models are created. It is non-trivial to decide which of them should be used for prediction in the final system. Obviously, if two models achieve the same accuracy on the training set, then it is reasonable to choose the simpler one. The question is how to characterize the complexity of machine learning models in a well-defined way.

The Vapnik–Chervonenkis dimension [Vapnik and Chervonenkis, 1971] is a widely accepted model complexity measure for sets of binary classifiers. The first part of this chapter will introduce the concept and show its utility in machine learning. Then, I will present known and new results about the Vapnik–Chervonenkis dimension of convex polyhedron classifiers.

3.1 Definitions

Recall that a binary classifier is an $\mathbb{R}^d \mapsto \{c_1, c_2\}$ mapping, where the values $c_1, c_2 \in \mathbb{R}$ are called labels. Without loss of generality assume that $c_1 = 1$ and $c_2 = 0$. Suppose that we have a set of binary classifiers $\mathcal{G}$ and a finite set of points $\mathcal{P} \subset \mathbb{R}^d$. We say that $\mathcal{G}$ *shatters* $\mathcal{P}$, if the elements of $\mathcal{P}$ can be arbitrarily labeled by the elements of $\mathcal{G}$.

For example, if $d = 1$, $\mathcal{G} = \{g_1(x) = \mathrm{th}(x-5), g_2(x) = -\mathrm{th}(x-3), g_3(x) = -\mathrm{th}(3-x), g_4(x) = -\mathrm{th}(5-x)\}$, and $\mathcal{P} = \{2, 4\}$, then $\mathcal{G}$ shatters $\mathcal{P}$, because classifier g_1 produces the labeling $(2 \to 0, 4 \to 0)$, g_2 produces $(2 \to 0, 4 \to 1)$, g_3 produces $(2 \to 1, 4 \to 0)$, and g_4 produces $(2 \to 1, 4 \to 1)$. If $\mathcal{G}$ is the same as before, and $\mathcal{P} = \{2, 4, 6\}$, then $\mathcal{G}$ does not shatter $\mathcal{P}$, because e.g. the labeling $(2 \to 0, 4 \to 1, 6 \to 0)$ cannot be obtained.

The *m-th shatter coefficient* of $\mathcal{G}$ denoted by $s(\mathcal{G}, m)$ is the maximum number of different labelings that can be produced with the members of $\mathcal{G}$ over m points. The 0-th shatter coefficient is defined as 1 for any $\mathcal{G}$. In the previous example $s(\mathcal{G}, 0) = 1$, $s(\mathcal{G}, 1) = 2$, and $s(\mathcal{G}, m) = 4$, if $m \geq 2$.

The *Vapnik–Chervonenkis (VC) dimension* of $\mathcal{G}$, denoted by $h(\mathcal{G})$, is the maximum number of points that can be shattered by $\mathcal{G}$. If $\mathcal{G}$ can shatter arbitrarily many points, then $h(\mathcal{G})$ is defined as ∞. With shatter coefficients $h(\mathcal{G})$ can be expressed as

$$h(\mathcal{G}) = \max\{m \in \mathbb{N} : s(\mathcal{G}, m) = 2^m\}.$$

The VC dimension of $\mathcal{G}$ in the previous example is 2, because there exist a 2-element point set that can be shattered by $\mathcal{G}$, but there is no such 3-element point set.

The concept was originally defined by Vapnik and Chervonenkis [Vapnik and Chervonenkis, 1971]. It can be viewed as a measure of complexity for binary classification models. The main utility of the concept is that we can obtain an upper bound on the error probability of a classifier on new examples with the help of it.

Given a binary classifier g, there is no general connection between its test error probability $L(g) = \mathbf{P}\{g(\mathbf{X}) \neq Y\}$ and its training error rate $\widehat{L}_n(g) = \frac{1}{n}\sum_{i=1}^{n} I\{g(\mathbf{x}_i) \neq y_i\}$. However, if we know a priori that $g \in \mathcal{G}$ and $h(\mathcal{G}) < \infty$, then with probability $1 - \delta$:

$$L(g) \leq \widehat{L}_n(g) + \sqrt{8\frac{h(\mathcal{G})\ln(2en/h(\mathcal{G})) + \ln(2/\delta)}{n}}. \tag{3.1}$$

A nice consequence of the above inequality is that if the VC dimension is finite, then we are able to bound the test error without using a test set. The proof of the theorem can be found e.g. in [Bousquet et al., 2004], along with other interesting results.

3.1.1 Convex polyhedron function classes

Recall that a function $g : \mathbb{R}^d \mapsto \{1, 0\}$ is a linear classifier, if it can be written as

$$g(\mathbf{x}) = \mathrm{th}(\mathbf{w}^T\mathbf{x} + b),$$

where $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are the parameters of the model. Let LIN_d denote the set of all d-dimensional linear classifiers.

We get a richer function class, if the label is decided by combining the outputs of K linear classifiers:

$$\begin{aligned} g(\mathbf{x}) &= f(l_1(\mathbf{x}), \ldots, l_K(\mathbf{x})), \\ l_k(\mathbf{x}) &= \mathrm{th}(\mathbf{w}_k^T\mathbf{x} + b_k), \quad k = 1, \ldots, K, \end{aligned}$$

where f is an arbitrary $\{1, 0\}^K \mapsto \{1, 0\}$ mapping called *composition function*. Classifiers that can be expressed in this form are called *polyhedron classifiers*. Let $\mathrm{POL}_{d,K}$ denote the set of all d-dimensional polyhedron classifiers using K linear decisions. The function class $\cup_{k=1}^{\infty}\mathrm{POL}(d, k)$ is quite extensive: among others it contains all d-dimensional linear classifiers, ID3 decision trees and nearest neighbor classifiers.

We obtain interesting subsets of $\mathrm{POL}_{d,K}$, if we restrict the composition function to minimum and maximum taking:

$$\begin{aligned} \mathrm{MIN}_{d,K} &= \{g : g \in \mathrm{POL}_{d,K}, f = \min\}, \\ \mathrm{MAX}_{d,K} &= \{g : g \in \mathrm{POL}_{d,K}, f = \max\}, \\ \mathrm{MINMAX}_{d,K} &= \{g : g \in \mathrm{POL}_{d,K}, f \in \{\min, \max\}\}. \end{aligned}$$

It is true for all of the above cases that the decision boundary is the surface of a d-dimensional convex K-polyhedron. The label of the inner (convex) region is always 1 in $\mathrm{MIN}_{d,K}$ always 0 in $\mathrm{MAX}_{d,K}$, and unrestricted in $\mathrm{MINMAX}_{d,K}$. Therefore, $\mathrm{MIN}_{d,K}$ and $\mathrm{MAX}_{d,K}$ refer to the case in which we know in advance what the label of the inner class is, and $\mathrm{MINMAX}_{d,K}$ to the general case of convex polyhedron classification. Note that the decision boundary belongs to the inner region in the case of $\mathrm{MIN}_{d,K}$ and to the outer region in the case of $\mathrm{MAX}_{d,K}$.

3.2 Known facts

In this section, we will overview some known facts about the VC dimension $\mathrm{MIN}_{d,K}$, $\mathrm{MAX}_{d,K}$, and $\mathrm{MINMAX}_{d,K}$. Clearly, $h(\mathrm{MIN}_{d,K}) = h(\mathrm{MAX}_{d,K})$, therefore it is enough to state theorems for $\mathrm{MIN}_{d,K}$ and $\mathrm{MINMAX}_{d,K}$.

Let us start with the simplest special cases. If $d = 1$ and $K = 1$, then $h(\mathrm{MIN}_{d,K}) = h(\mathrm{MINMAX}_{d,K}) = 2$. If $d = 1$ and $K > 1$, then $h(\mathrm{MIN}_{d,K}) = 2$ and $h(\mathrm{MINMAX}_{d,K}) = 3$. If $K = 1$, then $\mathrm{MIN}_{d,K}$, $\mathrm{MINMAX}_{d,K}$, and LIN_d are identical. It is known since long ago that $h(\mathrm{LIN}_d) = d + 1$. For showing this, we will use a simple lemma [Rodríguez, 2004].

Lemma 3.1. *Let $\mathcal{F}$ be a set of $\mathbb{R}^d \mapsto \mathbb{R}$ functions, and let $\mathcal{G}$ be the set of binary classifiers that can be written as $g(\mathbf{x}) = \mathrm{th}(f(\mathbf{x}))$, $f \in \mathcal{F}$. If $\mathcal{F}$ is an m-dimensional vector space, then $h(\mathcal{G}) \leq m$.*

Proof. Assume that we have $m + 1$ arbitrary points $\mathbf{p}_1, \ldots, \mathbf{p}_{m+1} \in \mathbb{R}^d$. Consider the linear transformation $\mathcal{L} : \mathcal{F} \mapsto \mathbb{R}^{m+1}$ defined by

$$\mathcal{L}(f) = [f(\mathbf{p}_1), \ldots, f(\mathbf{p}_{m+1})] .$$

Since the image space $\mathcal{LF}$ is of dimension m, there must exist a nonzero vector $\boldsymbol{\alpha} \in \mathbb{R}^m$ orthogonal to $\mathcal{LF}$. Thus, there exist scalars $\alpha_1, \ldots, \alpha_{m+1}$ not all zero such that

$$\sum_{i=1}^{m+1} \alpha_i f(\mathbf{p}_i) = 0, \quad \text{for all } f \in \mathcal{F}.$$

Separate the sum above according to whether α_i is negative ($i \in \mathcal{I}_-$) or not ($i \in \mathcal{I}_+$), to obtain

$$\sum_{i \in \mathcal{I}_+} \alpha_i f(\mathbf{i}_i) = \sum_{i \in \mathcal{I}_-} -\alpha_i f(\mathbf{p}_i).$$

Without loss of generality we can assume that $\mathcal{I}_-$ is not empty. $\mathcal{G}$ does not shatter $\mathcal{P}$, because it is impossible to obtain $f(\mathbf{p}_i) \geq 0$ for all $i \in \mathcal{I}_+$ and $f(\mathbf{p}_i) < 0$ for all $i \in \mathcal{I}_-$. (For this we should make the left hand side of the above equation greater or equal zero, and the right hand side less than zero.) □

Now we are ready to prove the classical result about $h(\mathrm{LIN}_d)$.

Theorem 3.1. $h(\mathrm{LIN}_d) = d + 1$.

Proof. Let us consider the point set $\mathcal{P} = \{\mathbf{p}_0, \ldots, \mathbf{p}_d\} \subset \mathbb{R}^d$, where $\mathbf{p}_0 = \mathbf{0}$, and $\mathbf{p}_i$ is the i-th unite vector, for $i \geq 1$. The system of linear equations $\mathbf{w}^T\mathbf{p}_i + b = y_i$ $(i = 0, \ldots, d)$ has a solution for every $y_0, \ldots, y_d \in \mathbb{R}$, therefore $\mathcal{P}$ can be arbitrarily labeled by the elements of LIN_d. This proves $h(\mathrm{LIN}_d) \geq d + 1$.

Let $\mathcal{F}$ denote the set of linear functions in d variables. $\mathcal{F}$ is $(d+1)$-dimensional vector space, and the elements of LIN_d can be written as $g(\mathbf{x}) = \mathrm{th}(f(\mathbf{x}))$. Applying the previous lemma proves $h(\mathrm{LIN}_d) \leq d + 1$. □

For $\mathrm{MIN}_{2,K}$, $\mathrm{MIN}_{3,K}$, and $\mathrm{MIN}_{3,4}$ we can find results in [Dobkin and Gunopulos, 1995].

Theorem 3.2. *If $K \geq 2$, then $h(\mathrm{MIN}_{2,K}) = 2K + 1$.*

Proof. If we place $2K + 1$ points into the vertices of a regular $(2K + 1)$-gon, then every labeling can be produced with K lines. This implies $h(\mathrm{MIN}_{2,K}) \geq 2K + 1$. For proving the upper bound, consider an arbitrary point set $\mathcal{P}$ of size $(2K+2)$. If one of the points is located inside the convex

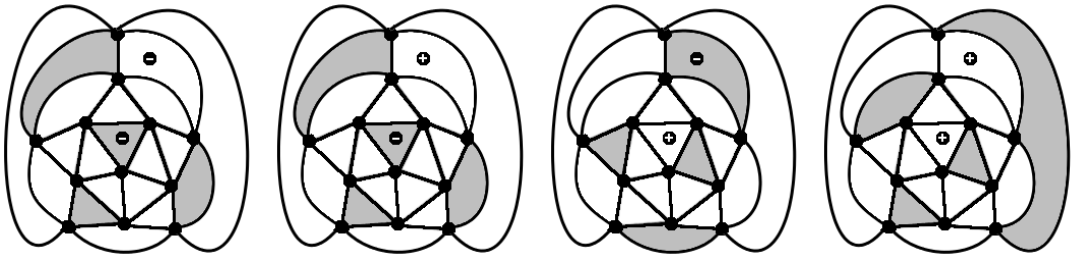


Figure 3.1: How to choose the independent faces based on the label of the extra points. Note that some faces are never chosen.

hull of the others, then $\mathcal{P}$ cannot be shattered by $\mathrm{MIN}_{2,K}$, because it is impossible to classify the inside point as 0 and the outside points as 1. If the convex hull of $\mathcal{P}$ is a convex $(2K+2)$-gon, then $\mathcal{P}$ cannot be shattered by $\mathrm{MIN}_{2,K}$ again, because it is impossible to obtain the alternating labeling with K lines. □

Theorem 3.3. $h(\mathrm{MIN}_{3,K}) \leq 4K$.

Proof. Let $\mathcal{P}$ be an arbitrary 3-dimensional point set of size $4K+1$. If one of the points is inside the convex hull of the others, then $\mathcal{P}$ cannot be shattered by $\mathrm{MIN}_{3,K}$, because it is impossible to label the outside points as 1 and the inside point as 0.

If the convex hull of $\mathcal{P}$ has $|\mathcal{P}|$ vertices, then let $\mathcal{A}$ be the vertex adjacency graph of the convex hull. Each vertex of $\mathcal{A}$ corresponds to an element of $\mathcal{P}$. $\mathcal{A}$ is a planar graph, therefore its vertices can be four colored [Appel and Haken, 1977]. As a consequence, $\mathcal{A}$ contains an independent set of vertices of size $K+1$. $\mathcal{P}$ cannot be shattered by $\mathrm{MIN}_{3,K}$, because it is impossible to label the independent points as 1 and the other points as 0 with K planes. □

Theorem 3.4. $h(\mathrm{MIN}(3,4)) \geq 14$.

Proof. Let us place the first 12 points into the vertices of a regular icosahedron. Clearly, these points can be shattered by $\mathrm{MIN}_{3,4}$. (The 4 planes are obtained from small perturbations of 4 independent faces.) Now place 2 extra points outside the the icosahedron but close to its surface, above 2 faces that are third neighbor of each other. Surprisingly, this larger point set can still be shattered by $\mathrm{MIN}_{3,4}$. This can be easily verified if we draw the planar graph of the icosahedron. The key is that the 4 independent faces are chosen based on the label of the 2 extra points (see Figure 3.1). □

In the general case it is possible to get a nice upper bound with combinatorial tools. The theorem we will prove now originally appeared in [Haussler and Welzl, 1987] (in a bit stronger form, but with a less detailed proof).

Theorem 3.5. $h(\mathrm{MIN}_{d,K}) < 2(d+1)K\log_2((d+1)K)$.

Proof. If $d = 1$, then the statement is true therefore we can assume $d \geq 2$. We will exploit that $\mathrm{MIN}_{d,K}$ classifiers consist of simple components that are combined in a simple way. Denote the m-th shatter coefficient of $\mathrm{MIN}_{d,K}$ by S_m and that of LIN_d by s_m. Given m points, each component classifier is able to produce at most s_m different labelings. This implies that

$$S_m \leq (s_m)^K.$$

From Sauer's lemma [Sauer, 1972] we know that

$$s_m \leq \sum_{i=0}^{d+1} \binom{m}{i}.$$

If $m \geq d+1$, then the sum of binomial coefficients can be bounded from above as

$$\sum_{i=0}^{d+1} \binom{m}{i} \leq \left(\frac{m}{d+1}\right)^{d+1} \left[\sum_{i=0}^{d+1} \binom{m}{i} \left(\frac{d+1}{m}\right)^{i}\right] \leq \left(\frac{m}{d+1}\right)^{d+1} \left(1+\frac{d+1}{m}\right)^{m} < \left(\frac{em}{d+1}\right)^{d+1}.$$

If $d \geq 2$, then

$$\left(\frac{em}{d+1}\right)^{d+1} < m^{d+1}.$$

Putting the inequalities together yields

$$S_m \leq (s_m)^K < m^{(d+1)K}.$$

Let us introduce the notation $x = (d+1)K$. If $m = 2x\log_2(x)$ and $x \geq 2$, then it can be shown that $m^{(d+1)K} \leq 2^m$. Rewriting m and $(d+1)K$ yields

$$[2x\log_2(x)]^x \leq \left[x^2\right]^x.$$

The above inequality is true for all $x \geq 2$, therefore $m^{(d+1)K} \leq 2^m$.

Moreover, it follows easily that $m^{(d+1)K} \leq 2^m$ is true also for $m = \lceil 2x\log_2(x)\rceil$, if $m \geq 2$. We got that the m-th shatter coefficient of MIN$_{d,K}$ is smaller than 2^m, if $m = \lceil 2(d+1)K\log_2((d+1)K)\rceil$. Therefore, $h(\text{MIN}_{d,K}) < 2(d+1)K\log_2((d+1)K)$. □

The straightforward way of obtaining results for $h(\text{MINMAX}_{d,K})$ is applying Assouad's lemma [Assouad, 1983]. The lemma states that for any function classes $\mathcal{F}$ and $\mathcal{G}$ with finite VC dimension, $h(\mathcal{F}\cup\mathcal{G}) \leq h(\mathcal{F}) + h(\mathcal{G}) + 1$. As a consequence, $h(\text{MINMAX}_{d,K}) \leq 2h(\text{MIN}_{d,K}) + 1$.

In the cases studied so far we get the following upper bounds:

- $h(\text{MINMAX}_{2,K}) \leq 4K + 3$, if $K \geq 2$,
- $h(\text{MINMAX}_{3,K}) \leq 8K + 1$,
- $h(\text{MINMAX}_{d,K}) \leq 4(d+1)K\log_2((d+1)K) + 1$.

3.3 The VC dimension of MINMAX$_{2,K}$

We have seen previously that $h(\text{MIN}_{2,K}) = 2K + 1$. Determining the Vapnik–Chervonenkis dimension of MINMAX$_{2,K}$ is a substantially harder problem. Now I will show that MINMAX$_{2,K} = 2K + 2$, if $K \geq 2$. I remark that the proof first appeared in [Takács, 2007].

Theorem 3.6.

$$h(\text{MINMAX}_{2,K}) = \begin{cases} 3 & \text{if } K = 1, \\ 2K+2 & \text{if } K \geq 2. \end{cases}$$

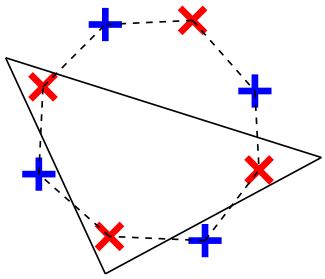


Figure 3.2: A point set in convex position. The red and a blue signs cannot be separated with a triangle, because for this we should intersect all edges of a convex 8-gon with 3 lines.

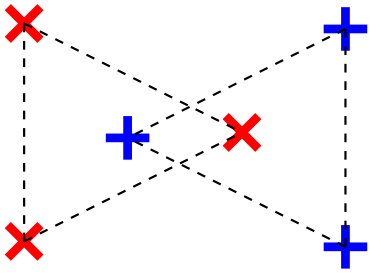


Figure 3.3: A point set in tangled position. The red and the blue signs can never be separated with a convex K-gon, regardless of the value of K.

3.3.1 Concepts for the proof

Definition 3.1. A planar point set is said to be in *convex position*, if its elements are the vertices of a convex polygon.

In other words, $\mathcal{P}$ does not have two distinct subsets $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that the convex hull of $\mathcal{Q}_1$ contains a point from $\mathcal{Q}_2$. The following simple properties will also be used in the proof:

- $\mathcal{P}$ is in convex position, if and only if every 4-element subset of $\mathcal{P}$ is in convex position.
- A 2-element subset of $\mathcal{P}$ is called an *edge*, if the line segment connecting the two points is an edge of the convex hull of $\mathcal{P}$. $\mathcal{P}$ is in convex position, if and only if every 5-element subset of $\mathcal{P}$ containing an edge is in convex position.

Note that $\text{MINMAX}_{2,K}$ cannot shatter $2K+2$ convexly positioned points, because for the alternating labeling we should intersect all edges of a convex $(2K+2)$-gon with K lines (see Figure 3.2). This is also true for $\text{MIN}_{2,K}$, moreover it implies that $h(\text{MIN}2, K) \leq 2K+1$, since $\text{MIN}_{2,K}$ can shatter only convexly positioned point sets. The main difference between the two function classes is that $\text{MINMAX}_{2,K}$ is able to shatter non-convexly positioned point sets too.

Definition 3.2. A planar point set $\mathcal{P}$ is said to be in *tangled position*, if it has two distinct subsets $\mathcal{Q}_1$ and $\mathcal{Q}_2$, such that the convex hull of $\mathcal{Q}_1$ contains a point from $\mathcal{Q}_2$ and the convex hull of $\mathcal{Q}_2$ contains a point from $\mathcal{Q}_1$.

$\mathcal{Q}_1$ and $\mathcal{Q}_2$ are called the *tangled subsets*. If $\mathcal{P}$ is not in tangled position, then it is said to be *tangle-free*. Note that for any K, a tangled set of points cannot be shattered by $\text{MINMAX}_{2,K}$, because it is impossible to separate $\mathcal{Q}_1$ from $\mathcal{Q}_2$ (see Figure 3.3).

3.3.2 The proof

The case $K = 1$ is trivial, therefore we consider only the case $K \geq 2$. We want to prove that $h(\text{MINMAX}_{2,K}) = 2K+2$. It is easy to see that $h(\text{MINMAX}_{2,K}) \geq 2K+2$. For this we should just place $2K+1$ points along a circle, in the vertices of a regular $(2K+1)$-gon and put an additional point in the center.

Consider an arbitrary labeling of these $2K+2$ points. We will refer to the points that have the same label as the center as red points while to the others as blue points. There can be at most K blue sequences along the circle. If the longest blue sequence is at most K long, then

each blue sequence can be separated from the red points with 1 line. If the length of the longest blue sequence is more than K, then that blue sequence can be separated from the red points with 2 lines, and each remaining one with 1 line. The number of the remaining blue sequences is not greater than $K-2$ (assuming $K \geq 2$). Therefore, K lines are always enough.

Proving the upper bound $h(\text{MINMAX}_{2,K}) \leq 2K+2$ is somewhat more difficult. We should show that no $2K+3$ points can be shattered by $\text{MINMAX}_{2,K}$. It suffices to consider point set in which no 3 points are co-linear. If there is a point set that can be shattered by $\text{MINMAX}_{2,K}$, then there also exists a generally positioned point set of the same size that can be shattered by $\text{MINMAX}_{2,K}$. (The second point set can be constructed from the first with small perturbations.)

Assume that $\text{MINMAX}_{2,K}$ shatters a generally positioned point set $\mathcal{P}$. So far we know two necessary conditions for this:

- $\mathcal{P}$ does not contain $2K+2$ points that are in convex position.
- $\mathcal{P}$ is tangle-free.

In the rest of the proof we will show that if $K \geq 2$ and $|\mathcal{P}| \geq 2K+3$, then these requirements are contradictory, therefore no $2K+3$ points can be shattered by $\text{MINMAX}_{2,K}$.

Theorem 3.7. *Let $\mathcal{P}$ be a planar point set in general position. If $\mathcal{P}$ is tangle-free and $|\mathcal{P}| \neq 6$, then $\mathcal{P}$ contains $|\mathcal{P}|-1$ points that are in convex position.*

Corollary 3.1. *If $|\mathcal{P}| = 2K+3$, and $K \geq 2$, then $\mathcal{P}$ is in tangled position or it contains $2K+2$ convexly positioned points. Therefore, $\mathcal{P}$ cannot be shattered by MINMAX$_{2,K}$.*

Proof. Denote the convex hull of $\mathcal{P}$ by $\text{conv}(\mathcal{P})$. If $\text{conv}(\mathcal{P})$ is a point or a line segment, then the statement is trivial. The other cases are not so easy, because we can put arbitrarily many points into $\text{conv}(\mathcal{P})$ such that the requirements of the theorem are fulfilled. Along the property of being a vertex of $\text{conv}(\mathcal{P})$ or not, the elements of $\mathcal{P}$ can be classified as *outside* or *inside* points.

At first consider the case when $\text{conv}(\mathcal{P})$ is a triangle. Denote the 3 outside points by A, B, and C. If $|\mathcal{P}| \leq 5$, then the statement of the theorem can be easily verified. Therefore we can assume that $|\mathcal{P}| \geq 7$, and we have at least 4 inside points.

Now select two arbitrary inside points and denote them by D and E. The line DE intersects two edges of the triangle ABC. Without loss of generality we can assume that the line DE intersects the edge AB in the direction of D and intersects the edge AC in the direction of E. Draw the following line segments into the triangle ABC:

- Segment DE, extended to the edges AB and AC,
- Segment BD, extended to the edge AC,
- Segment CE, extended to the edge AB,
- The extension of segment AD in the direction of D,
- The extension of segment AE in the direction of E,
- Segment BE,
- Segment CD.

These line segments partition the triangle ABC into 14 distinct regions $(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{14})$. Number them according to Figure 3.4. Now try to put a third inside point F into the triangle ABC without introducing a tangle.

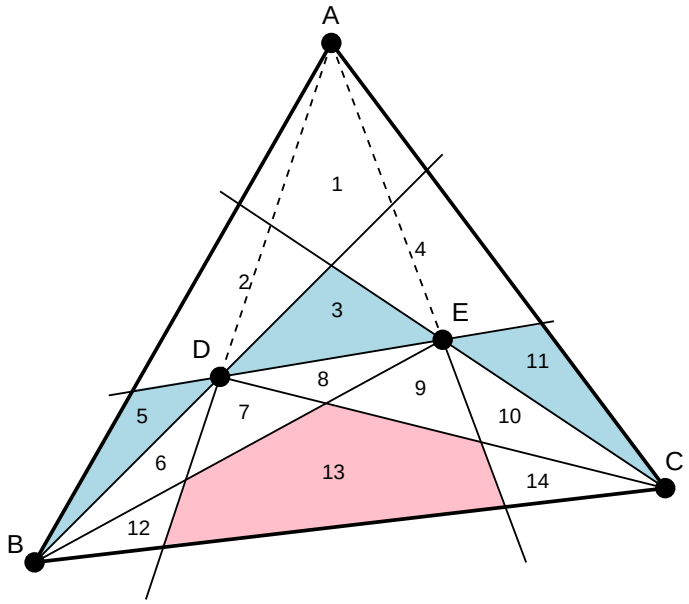


Figure 3.4: The 14 regions generated by placing two points into a triangle.

Lemma 3.2. *If $F \notin \mathcal{R}_3 \cup \mathcal{R}_5 \cup \mathcal{R}_{11} \cup \mathcal{R}_{13}$, then $\mathcal{P}$ is in tangled position.*

Proof.

- If $F \in \mathcal{R}_1 \cup \mathcal{R}_2$, then $\mathcal{Q}_1 = \{A, C, D\}$ and $\mathcal{Q}_2 = \{B, E, F\}$ are the tangled subsets.
- If $F \in \mathcal{R}_1 \cup \mathcal{R}_4$, then $\mathcal{Q}_1 = \{A, B, E\}$ and $\mathcal{Q}_2 = \{C, D, F\}$.
- If $F \in \mathcal{R}_6 \cup \mathcal{R}_7 \cup \mathcal{R}_8$, then $\mathcal{Q}_1 = \{B, D, E\}$ and $\mathcal{Q}_2 = \{A, C, F\}$.
- If $F \in \mathcal{R}_8 \cup \mathcal{R}_9 \cup \mathcal{R}_{10}$, then $\mathcal{Q}_1 = \{C, D, E\}$ and $\mathcal{Q}_2 = \{A, B, F\}$.
- If $F \in \mathcal{R}_6 \cup \mathcal{R}_{12}$, then $\mathcal{Q}_1 = \{B, C, D\}$ and $\mathcal{Q}_2 = \{A, E, F\}$.
- If $F \in \mathcal{R}_{10} \cup \mathcal{R}_{14}$, then $\mathcal{Q}_1 = \{B, C, E\}$ and $\mathcal{Q}_2 = \{A, D, F\}$.

□

Lemma 3.3. *If $F \in \mathcal{R}_{13}$, then $\mathcal{P}$ is in tangled position.*

Proof. If $F \in \mathcal{R}_{13}$, then lines DE, DF and EF partition the triangle ABC into 13 distinct regions $(\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{13})$. Number them according to Figure 3.5. Now try to place a 4th inside point G without introducing a tangle.

- If $G \in \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_4 \cup \mathcal{S}_5 \cup \mathcal{S}_6 \cup \mathcal{S}_{10} \cup \mathcal{S}_{11} \cup \mathcal{S}_{12} \cup \mathcal{S}_{13}$, then $\mathcal{P}$ is in tangled position by Lemma 3.2.
- If $G \in \mathcal{S}_3 \cup \mathcal{S}_8$, then $\mathcal{Q}_1 = \{A, D, E, F\}$ and $\mathcal{Q}_2 = \{B, C, G\}$ are the tangled subsets.
- If $G \in \mathcal{S}_7 \cup \mathcal{S}_8$, then $\mathcal{Q}_1 = \{B, D, E, F\}$ and $\mathcal{Q}_2 = \{A, C, G\}$.

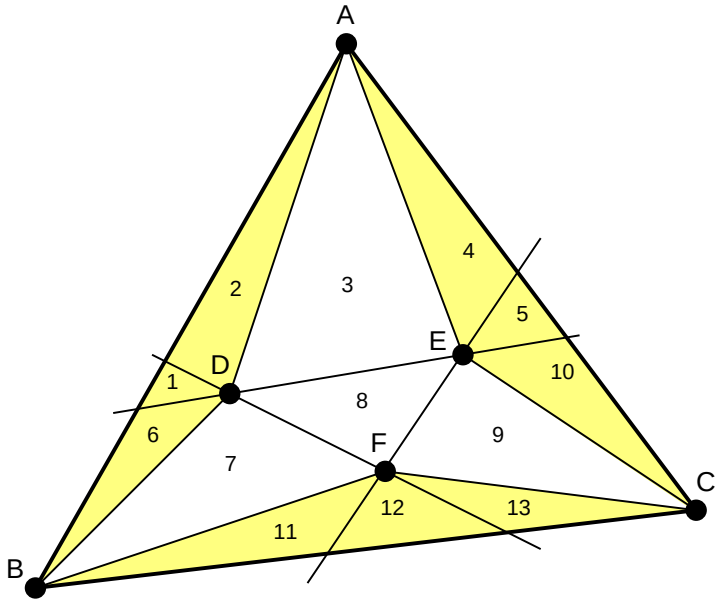


Figure 3.5: The 13 regions generated by placing the third point into $\mathcal{R}_{13}$.

- If $G \in \mathcal{S}_8 \cup \mathcal{S}_9$, then $\mathcal{Q}_1 = \{C, D, E, F\}$ and $\mathcal{Q}_2 = \{A, B, G\}$.

□

Remark. If F is a non-boundary point of $\mathcal{R}_{13}$, then $\{A, B, C, D, E, F\}$ is tangle-free, but has no 5-element subset in convex position. This is why the restriction $|\mathcal{P}| \neq 6$ had to be made. However, by Lemma 3.3 this arrangement is an irrelevant branch that cannot be continued.

The following fact is a simple consequence of Lemma 3.2 and Lemma 3.3:

Corollary 3.2. *If a set of two outside and three inside points is not in convex position, then then $\mathcal{P}$ is in tangled position.*

Now we are ready to finish the special case, when conv($\mathcal{P}$) is a triangle.

Lemma 3.4. *Let $\mathcal{P}$ be a planar point set in general position. If $\mathcal{P}$ is tangle-free, $|\mathcal{P}| \neq 6$ and conv($\mathcal{P}$) is a triangle, then we can select $|\mathcal{P}| - 1$ points from $\mathcal{P}$ that are in convex position.*

Proof. Let us analyze the situation after placing m inside points. Denote the union of $\{B, C\}$ and the first m inside points with $\mathcal{T}_m$. We know that $\mathcal{T}_2$ is in convex position. We will show that if $m \geq 2$, then the convex position of $\mathcal{T}_m$ implies the convex position of $\mathcal{T}_{m+1}$. To verify this assume indirectly that $\mathcal{T}_m$ is in convex position but $\mathcal{T}_{m+1}$ is not. Since $\{B, C\}$ is always an edge of conv($\mathcal{T}_{m+1}$) and $m \geq 2$, this means that $\mathcal{T}_{m+1}$ has a 5-element subset that contains $\{B, C\}$ and is not in convex position. Then by Corollary 3.2, $\mathcal{P}$ is in tangled position, which is a contradiction. Thus the convex position of $\mathcal{T}_{m+1}$ follows from the convex position of $\mathcal{T}_m$. As a consequence, the set $\mathcal{T}_{|\mathcal{P}|-3} = \mathcal{P} \setminus \{A\}$ is also in convex position. □

Remark. If we prohibit to place the third inside point into $\mathcal{R}_{13}$, then the condition $|\mathcal{P}| \neq 6$ can be omitted.

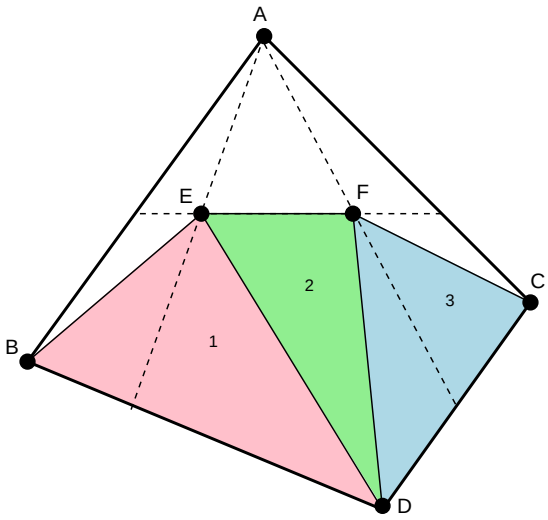


Figure 3.6: Regions in $BCDEF$, case I.

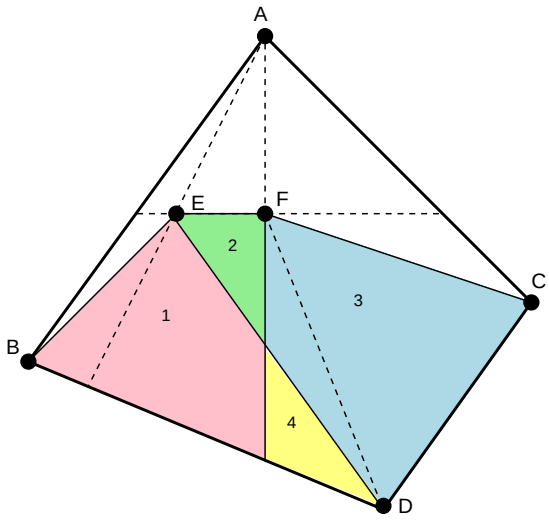


Figure 3.7: Regions in $BCDEF$, case II.

Now consider the case when $\text{conv}(\mathcal{P})$ is a quadrangle. Denote the 4 outside points with A, B, C and D. If $|\mathcal{P}| \leq 5$, then the statement is trivial, therefore we can assume that we have at least two inside points. Select two arbitrary inside points and denote them by E and F. The line EF intersects two adjacent edges of the quadrangle $ABCD$, because otherwise $\mathcal{P}$ would be in tangled position. Without loss of generality we can assume that the line EF intersects the edge AB in the direction of E and intersects the edge AC in the direction of F. Now try to place a third inside point G into the quadrangle $ABCD$.

Lemma 3.5. *If G is inside the pentagon $BCDEF$, then $\mathcal{P}$ is in tangled position.*

Proof. There are two possible cases:

1. The extension of AD in the direction of D and the extension of AE in the direction E intersect different edges of the quadrangle $ABCD$.

2. The extension of AD in the direction of D and the extension of AE in the direction E intersect the same edge of the quadrangle $ABCD$. We can assume without loss of generality that the intersected edge is BD.

In the first case, the line segments DE and DF partition the pentagon $BCDEF$ into 3 distinct regions ($\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$), as it can be seen in Figure 3.6. If $G \in \mathcal{R}_1 \cup \mathcal{R}_2$, then $\{B, D, E, F\}$ and $\{A, C, G\}$ are the tangled subsets. If $G \in \mathcal{R}_2 \cup \mathcal{R}_3$, then $\{C, D, E, F\}$ and $\{A, B, G\}$ are the tangled subsets.

In the second case, DE and the extension of AF in the direction of F partitions the pentagon $BCDEF$ into 4 distinct regions ($\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, $\mathcal{S}_4$), as it can be seen in Figure 3.7. If $G \in \mathcal{S}_1 \cup \mathcal{S}_2$, then $\{B, D, E, F\}$ and $\{A, C, G\}$ are the tangled subsets. If $G \in \mathcal{S}_2 \cup \mathcal{S}_3$, then $\{C, D, E, F\}$ and $\{A, B, G\}$ are the tangled subsets. If $G \in \mathcal{S}_4$, then $\{B, C, D, F\}$ and $\{A, E, G\}$ are the tangled subsets. □

By Lemma 3.5, inside points up from the third can be placed only into the region $ABC \setminus BCEF$ without introducing a tangle. This and Lemma 3.4 (applied to $\mathcal{P} \setminus \{D\}$) implies that

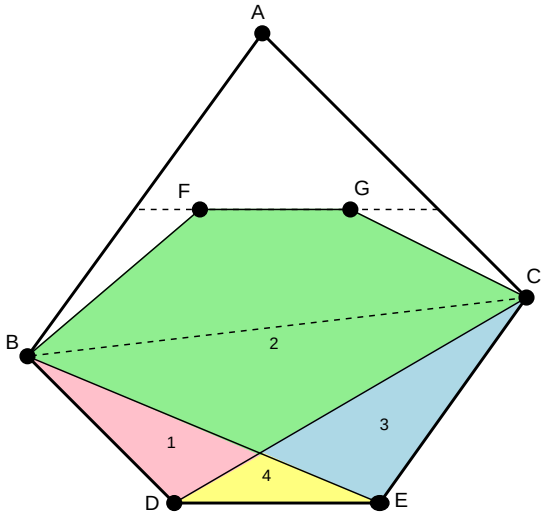


Figure 3.8: Regions in $BCDEFG$.

$\mathcal{P} \setminus \{A, D\}$ is in convex position. The restriction $|\mathcal{P} \setminus \{D\}| \neq 6$ can be now omitted, because the quadrangle $BCEF$ is a forbidden area. If $\mathcal{P} \setminus \{A, D\}$ is in convex position, then $\mathcal{P} \setminus \{A\}$ is too. This completes the proof of the special case when $\text{conv}(\mathcal{P})$ is a quadrangle.

Now consider the case when $\text{conv}(\mathcal{P})$ is a pentagon. Denote the 5 outside points by A, B, C, D and E. Using the same reasoning as before we can assume that we have at least two inside points. Pick two arbitrary inside points F and G. The line FG intersects two adjacent edges of the pentagon $ABCDE$, because otherwise $\mathcal{P}$ would be in tangled position. Without loss of generality assume that the line FG intersects the edge AB in the direction of F, intersects the edge AC in the direction of G, moreover BD and CE are edges of the pentagon $ABCDE$. Now try to place a third inside point H into the pentagon $ABCDE$.

Lemma 3.6. *If H is inside the hexagon $BCDEFG$, then $\mathcal{P}$ is in tangled position.*

Proof. Line segments BE and CD partition the hexagon $BCDEFG$ into 4 distinct regions $(\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4)$, as it can be seen in Figure 3.8. If $H \in \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$, then $\mathcal{P}$ is in tangled position by Lemma 3.5. If $H \in \mathcal{R}_4$, then $\mathcal{P}$ is tangled too, because there exists a line connecting two inside points that intersects non-adjacent edges of $\text{conv}(\mathcal{P})$. For example the line FH cannot intersect adjacent edges of $\text{conv}(\mathcal{P})$. □

By Lemma 3.6, inside points up from the third can be placed only into the region $ABC \setminus BCFG$ without introducing a tangle. This and Lemma 3.4 (applied to $\mathcal{P} \setminus \{D, E\}$) implies that $\mathcal{P} \setminus \{A, D, E\}$ and this wise $\mathcal{P} \setminus \{A\}$ is in convex position.

Finally consider the case when $\text{conv}(\mathcal{P})$ is a k-gon ($k \geq 6$). Denote the outside points by $A_1, A_2, \ldots, A_k$ and the inside points by $B_1, B_2, \ldots, B_m$. The line B_1B_2 intersects again two adjacent edges of $\text{conv}(\mathcal{P})$. Without loss of generality assume that the line B_1B_2 intersects the edges A_1A_k and A_1A_k. No inside point can be located in the $(k+1)$-gon $A_2A_3 \ldots A_kB_1B_2$, because otherwise $\mathcal{P}$ would be in tangled position by Lemma 3.6. But then it follows as before that $\{A_2, A_k\} \cup \{B_1, B_2, \ldots, B_m\} = \mathcal{P} \setminus A_1$ is in convex position. □

3.4 New lower bounds

This section contains new lower bounds on the Vapnik–Chervonenkis dimension of various convex polyhedron classifier types. The results presented here first appeared in [Takács and Pataki, 2007b].

Let us start with two easily obtainable lower bounds for $h(\mathrm{MIN}_{d,K})$ and $h(\mathrm{MINMAX}_{d,K})$.

Theorem 3.8 (sphere slicing)**.** *$h(\mathrm{MIN}_{d,K}) \geq dK$.*

*Proof.*We have to arrange dK points such that they can be shattered by $\mathrm{MIN}_{d,K}$. At first cut K distinct slices from a d-dimensional hypersphere with K hyperplanes. Denote the i-th hyperplane with $\mathcal{H}_i$ and denote the intersection of the hypersphere surface and the i-th hyperplane with $\mathcal{C}_i$.

Now define the point set $\mathcal{P}$ by assigning d different points in each $\mathcal{C}_i$. Points belonging to the same $\mathcal{C}_i$ can be shattered by one hyperplane, moreover, it can be required too, that the label of all the other points have to be 1 according to this hyperplane. (The i-th hyperplane of the MIN_d, K classifier is obtained from a small perturbation of $\mathcal{H}_i$.) Because of the requirement, the i-th hyperplane influences only the the labeling of the i-th group, therefore $\mathcal{P}$ can be arbitrarily labeled by the members of $\mathrm{MIN}_{d,K}$. □

Theorem 3.9. *$h(\mathrm{MINMAX}_{d,K}) \geq dK + 1$.*

Proof. Arrange the first dK points as in the proof of the previous theorem, but now put an additional point into the center of the hypersphere. The previous theorem implies that the first dK points can be shattered by both $\mathrm{MIN}_{d,K}$ and $\mathrm{MAX}_{d,K}$. In the first case, the label of the central point is always 1, in the second case it is always 0. This means, that the $dK + 1$ points can be arbitrarily labeled by the members of $\mathrm{MINMAX}_{d,K}$. (The label of the central point determines whether to use a $\mathrm{MIN}_{d,K}$ or a $\mathrm{MAX}_{d,K}$ classifier.) □

The previous proofs are so simple that $h(\mathrm{MIN}_{d,K}) \geq dK$ and $h(\mathrm{MINMAX}_{d,K}) \geq dK + 1$ will be referred as the basic lower bounds. The following theorem improves the basic lower bounds by one:

Theorem 3.10. *If $K \geq 2$, then $h(\mathrm{MIN}_{d,K}) \geq dK + 1$ and $h(\mathrm{MINMAX}_{d,K}) \geq dK + 2$.*

Proof. We begin with the proof of the first statement in the special case $d = 3$. Denote the coordinates by x, y, and z. Place the first 5 points $\mathbf{p}_1, \ldots, \mathbf{p}_5$ on the plane xy, into the vertices of an origin-centered regular pentagon. On the plane xy these points can be shattered by 2 lines, moreover, the following constraints can be satisfied too:

- The lines have to be parallel.
- The lines cannot be axis-parallel.
- For any labeling, there exists an $\varepsilon_0 > 0$ such that the circle of radius ε_0 around the origin must belong to the inner region.

Place the next 2 points into $\mathbf{p}_6 = [1 \quad 0 \quad 1]$ and $\mathbf{p}_7 = [-1 \quad 0 \quad 1]$. Now we show that the first 7 points can be arbitrarily labeled by the members of $\mathrm{MIN}_{3,2}$, even if we require that an ε_0-ball around the origin have to belong to the inner region.

Consider an arbitrary labeling $\mathbf{p}_1 \to y_1, \ldots, \mathbf{p}_7 \to y_7$ that we want to obtain with a member of $\mathrm{MIN}_{3,2}$. Recall that a $\mathrm{MIN}_{3,2}$ classifier can be given by 8 parameters: $(w_{11}, w_{12}, w_{13}, b_1)$ and $(w_{21}, w_{22}, w_{32}, b_2)$.

If the input is one of the first 5 points, then the answer of the classifier is fully determined by the parameters (w_{11}, w_{12}, b_1) and (w_{21}, w_{22}, b_2). Adjust these parameters such that the classifier labels the first 5 points as desired. This can be done, moreover the following constraints can be satisfied too ($i = 1, 2$):

- $w_{2i} = -w_{1i} \neq 0$ (the lines are parallel but not axis-parallel),
- $\forall \delta_1^2 + \delta_2^2 \leq \varepsilon_0^2 : \min_i\{\delta_1 w_{i1} + \delta_2 w_{i2} + b_i\} \geq 0$ (the ball of radius ε_0 around the origin belongs to the inner region).

Based on y_6 and y_7 there are 4 possible cases ($i = 1, 2$):

- If $y_6 = y_7 = 1$, then the desired labeling can be obtained by $w_{i3} \gg 0$.
- If $y_6 = y_7 = 0$, then the desired labeling can be obtained by $w_{i3} \ll 0$.
- If $y_6 = 1, y_7 = 0$, then the desired labeling can be obtained by $w_{i3} = -w_{i1} - b_i + |w_{i1}|$.
- If $y_6 = 0, y_7 = 1$, then the desired labeling can be obtained by $w_{i3} = +w_{i1} - b_i + |w_{i1}|$.

If we choose a sufficiently small ε, then the ε-ball around the origin belongs to the inner region in all of the 4 cases.

Now define a sphere surface that encloses the 7 points arranged so far and passes through the point $[0 \quad 0 \quad -\varepsilon/2]$. On this surface there exist a segment, that belongs to the inner region at each labeling of the first 7 points. Let us place the remaining $3(K-2)$ points onto this subsurface in 3-element groups with the sphere slicing method. Each group can be shattered by one plane, moreover it can be assured, that the planes does not affect the labeling of the other groups and the first 7 points. Therefore the $7 + 3(K-2) = 3K + 1$ points can be shattered by $\mathrm{MIN}_{3,K}$.

The proof of the d-dimensional case is analogous with the 3-dimensional one. Now the first 5 points are placed onto the plane of the first 2 coordinates. The next $2(d-2)$ points are arranged in the following way:

$$\begin{aligned}
\mathbf{p}_6 &= [+1 \quad 0 \quad 1 \quad 0 \quad 0 \quad \ldots \quad 0], \\
\mathbf{p}_7 &= [-1 \quad 0 \quad 1 \quad 0 \quad 0 \quad \ldots \quad 0], \\
\mathbf{p}_6 &= [+1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \ldots \quad 0], \\
\mathbf{p}_9 &= [-1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \ldots \quad 0], \\
&\vdots \\
\mathbf{p}_{2d} &= [+1 \quad 0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 1], \\
\mathbf{p}_{2d+1} &= [-1 \quad 0 \quad 0 \quad 0 \quad \ldots \quad 0 \quad 1].
\end{aligned}$$

It can be shown (exactly the same way as in the case $d = 3$) that the first $2d + 1$ points can be shattered by $\mathrm{MIN}_{d,2}$, moreover it can be required too that an ε-ball around the origin have to belong to the inner region at each labeling. Then a hypersphere surface is defined that encloses the first $2d + 1$ points and passes through the point $[0 \quad 0 \quad -\varepsilon/2 \quad \ldots \quad -\varepsilon/2]$. There exist a segment on this surface that belongs to the inner region at each labeling. The remaining $d(K-2)$ points are placed onto this subsurface with the sphere slicing method.

The statement on $\mathrm{MINMAX}_{d,K}$ can be proved with the same construction with an additional point in the origin. The label of this point determines whether to use a $\mathrm{MIN}_{d,K}$ or a $\mathrm{MAX}_{d,K}$ classifier. □

The bound $h(\mathrm{MIN}_{d,K}) \geq dK + 1$ is not tight, if $d > 2$. For example, in the case $d = 3, K = 4$ it states that $h(\mathrm{MIN}_{3,4}) \geq 13$ We have seen previously that even $h(\mathrm{MIN}_{3,4}) \geq 14$ can be proved with the help of an icosahedron-based arrangement [Dobkin and Gunopulos, 1995]. This better bound can easily be extended to the case $d = 3, K > 4$.

Theorem 3.11. *If $K > 4$, then $h(\mathrm{MIN}_{3,K}) \geq 3K + 2$ and $h(\mathrm{MINMAX}_{3,K}) \geq 3K + 3$.*

Proof. Place the first 14 points exactly as in the proof of Theorem 3.4. From the proof of Theorem 3.4 we know that these points can be shattered by $\mathrm{MIN}_{3,4}$, and there exist a face of the icosahedron that is never selected. This means that we can define a sphere surface that encloses the first 14 points and has a segment that is always in the inner region. If we place the remaining $3(K-4)$ points on this subsurface with the sphere slicing method, then this point set of size $14 + 3(K-4) = 3K + 2$ can be shattered by $\mathrm{MIN}_{3,K}$.

The second statement of the theorem can be proved the same way. The only difference is that if we can use $\mathrm{MAX}_{3,K}$ classifiers too, then we can put an additional point into the center of the icosahedron. □

We got that in $\mathbb{R}^3$ the basic lower bounds can be improved by 2. With a more sophisticated version of the icosahedron trick it is possible to improve the basic lower bounds by 4 in $\mathbb{R}^4$.

Theorem 3.12. *If $K \geq 30$, then $h(\mathrm{MIN}_{4,K}) \geq 4K + 4$ and $h(\mathrm{MINMAX}_{4,K}) \geq 4K + 5$.*

Proof. Let us consider a *600-cell*, which is a finite regular 4-dimensional polytope, containing 600 tetrahedral cells (with 5 to an edge), 1200 triangular faces, 720 edges, and 120 vertices. The 600-cell is also called *hypericosahedron*, because it can be viewed as the 4-dimensional analog of the 3-dimensional icosahedron.

The vertices of an origin-centered 600-cell with edges of length $1/\phi$ (where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio) can be given as follows:

- 16 vertices of the form $[\pm\frac{1}{2} \quad \pm\frac{1}{2} \quad \pm\frac{1}{2} \quad \pm\frac{1}{2}]$.
- The 8 possible permutations of $[\pm 1 \quad 0 \quad 0 \quad 0]$.
- 96 vertices, obtained from the even permutations of $[\pm\frac{1}{2} \quad \pm\frac{1}{2}\phi \quad \pm\frac{1}{2}\phi^{-1} \quad 0]$.

The topological structure of the 600-cell is a system of subsets over the 120 vertices that gives which k vertices form a k-facet ($k = 2, 3, 4$; 2-facets are called edges, 3-facets are called faces and 4-facets are called cells). The topological structure of the 600-cell can be computed easily. At first we should generate the coordinates of the 120 vertices according to the previous scheme. Then we should identify the k-facets by examining every possible k vertices an checking whether they are at distance $1/\phi$ from each other not. The vertex adjacency graph of the 600-cell can be seen in Figure 3.9.

We say that 2 cells are adjacent, if they have at least 1 common vertex and 2 cells are independent, if they have no common vertices. The cell adjacency graph of the 600-cell can be easily obtained from its topological structure.

Remember that in the 3-dimensional case we tried to cover the vertices of the icosahedron with independent facets in many different ways. Now we want to cover the vertices of the 600-cell with independent cells in many possible ways. This means that we want to find many independent points in the cell adjacency graph.

With a simple program performing brute force computation, it is possible to find 1920 different coverings.[1] These coverings can be represented as a 1920-by-600 binary matrix $\mathbf{C}$ so that the element at position (i, j) is 1 if the i-th covering contains the j-th cell and 0 otherwise.

Now let us turn back to the statement $h(\mathrm{MIN}_{4,K}) \geq 4K + 4$, if $K \geq 30$. Place the first 120 points into the vertices of a 600-cell. These points can be shattered by $\mathrm{MIN}_{4,30}$, because it is

[1] The topological structure of the 600-cell, the cell adjacency graph and the coverings can be found at `http://www.sze.hu/~gtakacs/600cell.html`. It is difficult to verify these results by hand but it is easy to write a program that performs this.

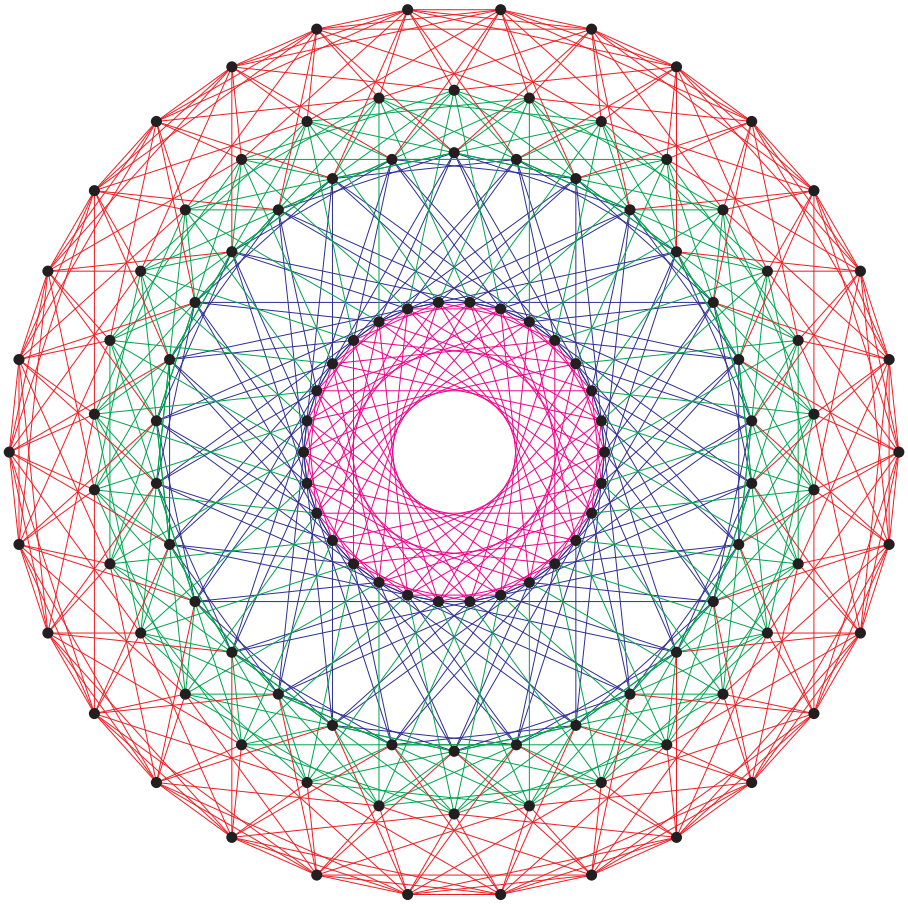


Figure 3.9: The vertex adjacency graph of the 600-cell.

possible to cover the 120 vertices with 30 independent cells. (The different labelings are obtained from small perturbations of the 30 independent cells.)

If there exist an L-column submatrix in the covering matrix $\mathbf{C}$ that has 2^L different rows, then L extra points can be placed into the arrangement such that the point set can still be shattered by $\mathrm{MIN}_{4,30}$. With a simple program it is possible to count the number of different rows in every L-column submatrices of $\mathbf{C}$. The largest L for that an appropriate subset of columns could be selected was $L = 4$. This means that it is possible to arrange $120 + 4 = 124$ points in $\mathbb{R}^4$ such that they can be shattered by $\mathrm{MIN}_{4,30}$.

Like in the previous theorems, we can define a 4-dimensional sphere that encloses the first 124 points and contains a surface segment that belongs to the inner region at each labeling of the first 124 points. If the remaining $4(K-30)$ points are placed onto this surface segment with the sphere slicing method, then the resulted $(dK+4)$-element point set can be arbitrarily labeled by $\mathrm{MIN}_{4,K}$ (assuming that $K \geq 30$). In the case of $\mathrm{MINMAX}_{4,K}$ the construction is the same, except that an additional point can be placed into the origin too. □

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

Albert Einstein

4 Applications

This chapter contains experiments demonstrating the utility of the algorithms proposed in the thesis. The organization of the chapter is the following: The first part will be about determining the linear and convex separability of point sets. The second part will deal with convex polyhedron methods for classification. The third part will be about convex polyhedron and other methods for collaborative filtering.

4.1 Determining linear and convex separability

4.1.1 Datasets

The datasets involved in the experiments were the following:

- **VOTES**: This dataset is part of the UCI (University of California, Irvine) machine learning repository [Asuncion and Newman, 2007], which is a well known collection of benchmark problems for testing machine learning algorithms. The dataset includes votes for each of the U.S. House of Representatives Congressmen on the $d = 16$ key votes identified by the Congressional Quarterly Almanac (98th Congress, 2nd session, 1984). The Congressional Quarterly Almanac lists nine different types of votes: voted for, paired for, and announced for (these three are encoded as 1), voted against, paired against, and announced against (these three are encoded as -1), voted present, voted present to avoid conflict of interest, and did not vote or otherwise make a position known (these three are encoded as 0). The dataset contains $M = 2$ classes (democrat or republican) and $n = 435$ examples (267 democrat, 168 republican).

- **WISCONSIN**: This dataset is also part of the UCI machine learning repository. It was originally obtained from the University of Wisconsin Hospitals, Madison, Wisconsin. The task to solve is to determine the benignness or malignancy ($M = 2$) of tumors based on $d = 9$ features extracted from mammograms. After removing examples with missing feature values the number of examples is $n = 683$ (444 benign, 239 malignant).

- **MNIST28**: This dataset is the training set part of the MNIST handwritten digit recognition database [LeCun and Cortes, 1999]. The $d = 784$ features represent pixel intensities of 28×28 sized images. The classes are associated with the digits, therefore the number of classes is $M = 10$. The number of examples is $n = 60000$ (5923 zeros, 6742 ones, 5958 twos, 6131 threes, 5842 fours, 5421 fives, 5918 sixes, 6265 sevens, 5851 eights, 5949 nines). Figure 4.1 shows some example images from the database. The dataset is sparse, the average frequency of 0 as a feature value is 80.9 %.

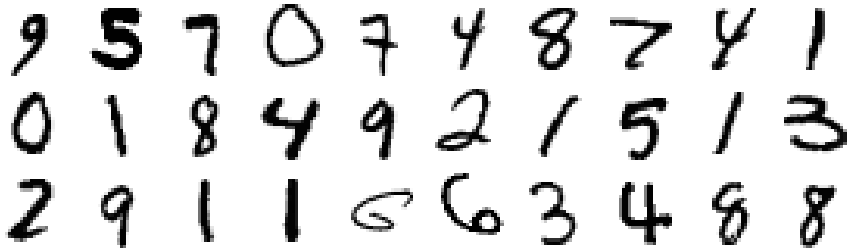


Figure 4.1: Examples from the MNIST28 database.

- **MNIST14**: This dataset was extracted from the MNIST28 by reducing image size to 14×14 pixels. Intensity values of the 14×14 images were obtained by averaging intensities in the corresponding 2×2 part of the original image. The average frequency 0 as a feature value is 74.4 %
- **MNIST7**: This dataset was extracted from the MNIST28 by reducing image size to 7×7 pixels. Intensity values of the 7×7 images were obtained by averaging intensities in the corresponding 4×4 part of the original image. The average frequency of 0 as a feature value is 62.3 %
- **MNIST4**: This dataset was extracted from the MNIST28 by reducing image size to 4×4 pixels. Intensity values of the 4×4 images were obtained by averaging intensities in the corresponding 7×7 part of the original image. The average frequency of 0 as a feature value is 41.3 %

If we want to refer to the subset of an MNIST dataset containing only classes A and B, then we will use postfix $/AB$ (e.g. MNIST04/49).

4.1.2 Algorithms

This subsection overviews the algorithms that were used in the experiments. For determining linear separability the following basic algorithms were tested:

- **LSEP**$_1$: The most straightforward, linear programming based method. It tries to find an arbitrary separating hyperplane (see page 36 for the details).
- **LSEP**$_1^*$: The dual formulation of LSEP$_1$ (see page 37).
- **LSEP**$_1^+$: A modified version of LSEP$_1^*$ (see page 37).
- **LSEP**$_2$: An alternative linear programming based method that tries to find a solution with small norm (see page 37).
- **LSEP**$_2^*$: The dual formulation of LSEP$_2$ (see page 38).
- **LSEP**$_S$: The support vector machine based method (see page 38). In the experiments it was used with setting $C = 10^6$.

All of these algorithms try to answer the question whether two point sets are linearly separable or not. The primal based methods (LSEP$_1$, LSEP$_2$) are constructive in the sense that they also

provide a separating hyperplane, if the answer is yes. The dual based methods (LSEP_1^*, LSEP_1^+ and LSEP_2^*) cannot do this, but they are able to output an easily verifiable proof, if the answer is no. The parameter ε was always set to 0.001 in the experiments.

The support vector based approach (LSEP_S) can only be considered as an approximate method, because C cannot be set to ∞, therefore the answer is not perfectly reliable in the nonseparable case.

For determining linear separability the following enhanced algorithms were tested:

- **LSEPX**$_1$, **LSEPX**$_2$: The first proposed method (see page 38). It tries to solve the problem incrementally. The index indicates which basic method (LSEP_1, LSEP_2) is used for solving subproblems.

- **LSEPY**$_1$, **LSEPY**$_2$: The modified version of the previous method (see page 39). It tries to further reduce running time by removing points from the active sets.

- **LSEPZ**: A dual based alternative of the previous two approaches (see page 39). It uses LSEP_1^+ for solving subproblems. It can be able to reduce the number of features in the linearly separable case.

- **LSEPZX**$_1$, **LSEPZX**$_2$, **LSEPZY**$_1$, **LSEPZY**$_2$: A hybrid method that tries to reduce the number of features with LSEPZ first, and then runs LSEPX_1, LSEPX_2, LSEPY_1, or LSEPY_2 on the reduced dataset (see page 40).

The variants of LSEPX, LSEPY, LSEPZX and LSEPZY are constructive methods, while LSEPZ is not. LSEPX and LSEPY was always run with setting $\gamma_k \equiv d$. The parameter ε was always set to 0.001 and ε_2 to 0.002.

For determining convex separability the following algorithms were tested:

- **CSEP**$_{X2}$, **CSEP**$_{Y2}$, **CSEP**$_{ZX2}$, **CSEP**$_{ZY2}$: Straightforward, linear programming based approach that tries to separate each outer point from the inner set individually (see page 40). The indices indicate which primal based method is used for the individual linear separations.

- **CSEP**$_2^*$, **CSEP**$_Z$: The dual based version of the the previous approach (see page 40). The indices indicate which dual based method is used for the individual linear separations.

- **CSEPC**: Proposed method for approximate convex separation (see page 41). It considers the lines connecting the centroid of the inner set with the outer points, and places hyperplanes perpendicular to these lines.

- **CSEPX**$_{X2}$, **CSEPX**$_{Y2}$, **CSEPX**$_{ZX2}$, **CSEPX**$_{ZY2}$, **CSEPX**$_2^*$, **CSEPX**$_Z$: Proposed method for fast and exact convex separation (see page 41). At first it tries to reduce the size of the outer set by applying CSEPC, and then it runs a variant of CSEP on the reduced dataset. The indices indicate which version of CSEP is used in the second phase.

All algorithms were implemented in Python [Rossum, 2006], using the numerical (NumPy) [Ascher et al., 2001] and the scientific (SciPy) [Jones et al., 2001] modules. The internal linear programming solver was the primal simplex method of the GNU Linear Programming Kit (GLPK) [Makhorin, 2009]. The reasons behind this choice were the following:

- GLPK is not the most sophisticated library for linear programming, but it is free, very well documented, and easy to use.

- The primal simplex method is a bit old fashioned and slow, but it is suitable for our purposes. The main goal is not to achieve the lowest possible running times but to compare different separation approaches.
- I did some experiments on small problems with the built-in interior point method of GLPK, and also with an own interior point solver written from scratch, but the running times were not promising. (However it is likely that a sophisticated interior point solver would beat the primal simplex method on large problems.)

GLPK was accessed from Python via the PyGLPK interface [Finley, 2008]. The internal support vector machine implementation was libsvm [Chang and Lin, 2001]. The hardware environment was a notebook PC with Intel Pentium M 2 GHz CPU and 1 Gb memory.

4.1.3 Types of separability

As we have seen previously, if we have point sets $\mathcal{P}$ and $\mathcal{Q}$, then it is possible to define different types of separability between them:

- S_0: $\mathcal{P}$ and $\mathcal{Q}$ are (convexly) inseparable, if $\mathcal{P} \cap \text{conv}(\mathcal{Q}) \neq \emptyset$ and $\mathcal{Q} \cap \text{conv}(\mathcal{P}) \neq \emptyset$.
- S_1: $\mathcal{P}$ and $\mathcal{Q}$ are convexly separable, if $\mathcal{P} \cap \text{conv}(\mathcal{Q}) = \emptyset$ or $\mathcal{Q} \cap \text{conv}(\mathcal{P}) = \emptyset$.
- S_2: $\mathcal{P}$ and $\mathcal{Q}$ are mutually convexly separable, if $\mathcal{P} \cap \text{conv}(\mathcal{Q}) = \emptyset$ and $\mathcal{Q} \cap \text{conv}(\mathcal{P}) = \emptyset$.
- S_3: $\mathcal{P}$ and $\mathcal{Q}$ are linearly separable, if $\text{conv}(\mathcal{P}) \cap \text{conv}(\mathcal{P}) = \emptyset$.

Note that S_3 implies S_2, and S_2 implies S_1, therefore S_3 can be considered as the strongest and S_1 as the weakest type of separability. Also observe that S_1 is the opposite of S_0.

In the following experiments the type of separability is determined for each pair of classes in the given datasets. At first, linear separability was checked with the LSEPZ method. If the point sets were not linearly separable, then also convex separability was checked with the CSEPX$_{X2}$ method.

The type of separability in the VOTES and the WISCONSIN dataset turned out to be S_1. The results for the MNIST28 dataset can be seen in Table 4.1, for the MNIST 14 dataset in Table 4.2, for the MNIST7 dataset in Table 4.3, and for the MNIST4 dataset in Table 4.4.

In the case of the MNIST28 dataset 38 of the subproblems are S_3- and 7 are S_2-type. This means that the majority of the 2-class subproblems are linearly separable, therefore it would be reasonable to use simple models, if we wanted to build classifiers.

If we reduce the size of the images, then it is natural to expect that the subproblems will become less separable. In the case of the MNIST14 dataset 18 of the subproblems are S_3- and 27 are S_2-type. In the case of the MNIST7 dataset 2 of the subproblems are S_3- and 43 are S_2-type. In the case of the MNIST4 dataset 2 of the subproblems are S_2-, 3 are S_1- and 40 are S_0-type. An interesting observation is that S_2 (mutual convex separability) occurs more frequently than S_1 (non-mutual convex separability).

If two point set are not convexly separable (or not mutually convexly separable), then it is natural to ask that how far they are from convex separability (mutual convex separability). A possible measure of inseparability is $\mathcal{I}_{\mathcal{PQ}} = |\mathcal{P} \cap \text{conv}(\mathcal{Q})|$, the number of points from $\mathcal{P}$ that are contained in the convex hull $\mathcal{Q}$.

The original versions of the presented convex separability algorithms are not able to determine the value of $\mathcal{I}_{\mathcal{PQ}}$, because they exit when the first inseparable outer point is found. However, it is trivial to modify the algorithms so that they calculate $\mathcal{I}_{\mathcal{PQ}}$. We just have to introduce a counter for the inseparable outer points and not exit when we find one.

Class 1	Class 2								
	1	2	3	4	5	6	7	8	9
0	S_3	S_3	S_3	S_3	S_3	S_3	S_3	S_3	S_3
1		S_3	S_3	S_3	S_3	S_3	S_3	S_3	S_3
2			S_2	S_3	S_3	S_3	S_3	S_2	S_3
3				S_3	S_2	S_3	S_3	S_2	S_3
4					S_3	S_3	S_3	S_3	S_2
5						S_3	S_3	S_2	S_3
6							S_3	S_3	S_3
7								S_3	S_2
8									S_3

Table 4.1: Types of separability in the MNIST28 dataset.

Class 1	Class 2								
	1	2	3	4	5	6	7	8	9
0	S_3	S_2	S_2	S_3	S_2	S_2	S_3	S_2	S_2
1		S_2	S_2	S_3	S_3	S_3	S_3	S_2	S_2
2			S_2	S_3	S_3	S_3	S_3	S_2	S_3
3				S_2	S_2	S_2	S_2	S_2	S_2
4					S_3	S_3	S_3	S_3	S_2
5						S_2	S_2	S_2	S_2
6							S_3	S_2	S_3
7								S_2	S_2
8									S_2

Table 4.2: Types of separability in the MNIST14 dataset.

Class 1	Class 2								
	1	2	3	4	5	6	7	8	9
0	S_3	S_2	S_2	S_2	S_2	S_2	S_2	S_2	S_2
1		S_2	S_2	S_2	S_2	S_2	S_2	S_2	S_2
2			S_2	S_2	S_2	S_2	S_2	S_2	S_2
3				S_2	S_2	S_2	S_2	S_2	S_2
4					S_2	S_2	S_2	S_2	S_2
5						S_2	S_2	S_2	S_2
6							S_3	S_2	S_2
7								S_2	S_2
8									S_2

Table 4.3: Types of separability in the MNIST7 dataset.

Class 1	Class 2								
	1	2	3	4	5	6	7	8	9
0	S_0	S_0	S_0	S_1	S_0	S_0	S_1	S_0	S_0
1		S_0	S_0	S_0	S_0	S_0	S_0	S_0	S_0
2			S_0	S_1	S_0	S_0	S_0	S_0	S_0
3				S_0	S_0	S_2	S_0	S_0	S_0
4					S_0	S_0	S_0	S_0	S_0
5						S_0	S_0	S_0	S_0
6							S_2	S_0	S_0
7								S_0	S_0
8									S_0

Table 4.4: Types of separability in the MNIST4 dataset.

Class 1	Class 2									
	0	1	2	3	4	5	6	7	8	9
0	5923	57	36	67	3	110	22	0	389	21
1	36	6742	43	228	14	21	14	126	260	109
2	1	6	5958	34	0	3	3	4	16	2
3	1	74	151	6131	7	65	0	119	41	58
4	0	13	15	4	5842	27	17	102	9	490
5	1	8	1	34	4	5421	5	11	27	13
6	5	7	22	0	7	8	5918	0	13	2
7	1	17	7	44	50	1	0	6265	4	372
8	273	310	162	498	2	458	9	69	5851	160
9	2	66	5	129	933	42	3	945	65	5949

Table 4.5: Number of examples from Class 1 contained in the convex hull of Class 2 in the MNIST4 dataset.

The $\mathcal{I}_{\mathcal{PQ}}$ values for the MNIST4 dataset calculated by modified CSEPX_{X2} are shown in Table 4.5. It is true to a certain degree that digit pairs considered to be more similar by humans have higher values. For example, there are 490 fours in the convex hull of nines and 933 nines in the convex hull of fours, but only 1 seven in the convex hull of zeros and 0 zeros in the convex hull of sevens.

4.1.4 Running times

Linear separability

The running times of basic methods for determining linear separability can be seen in Table 4.6. We get a measure of relative efficiency, if we rank the algorithms by running time for each problem, and then calculate the average rank of each algorithm [1]. According to this measure the order of basic methods on the given problems is the following: 1. LSEP_2^* (1.95), 2. LSEP_1^* (2.2), 3. LSEP_1^+ (2.45), 4. LSEP_1 (4.5), 5. LSEP_S (4.6), 6. LSEP_2 (5.3). The first 3 methods are the dual based ones. The fastest primal based method was LSEP_1.

It is interesting that sometimes LSEP_S was the fastest method, however its performance was not very stable. The running time of LSEP_S was more than 1800 seconds in every linearly nonseparable case (and even in some linearly separable ones).

The running times of the proposed incremental algorithms (LSEPX, LSEPY, and LSEPZ) are shown in Table 4.7. The order of the methods according to the average rank measure is: 1. LSEPX_2 (2.3), 2. LSEPY_2 (2.5), 3. LSEPX_1 (2.6), 4. LSEPY_1 (3.6), 5. LSEPZ (4.2).

If we consider the average running time instead of the average rank, then the superiority of LSEPX_2 and LSEPY_2 is even stronger. For example, on the MNIST14/01 dataset LSEPX_2 was 12 times faster than LSEPX_1, and LSEPY_2 was 17 times faster than LSEPY_1. Recall that among the basic methods LSEP_2 was always slower than LSEP_1 (since LSEP_2 tries to find a

[1] If there is a tie from position A to position B, then the rank is considered as $(A + B)/2$.

Problem		Running time (seconds)					
		LSEP_1	LSEP_1^*	LSEP_1^+	LSEP_2	LSEP_2^*	LSEP_S
VOTES	S_1	0.089	0.035	0.039	0.134	0.034	>1800
WISCONSIN	S_1	0.177	0.033	0.034	0.228	0.032	>1800
MNIST04/24	S_1	122.0	1.01	0.89	135.9	1.03	>1800
MNIST04/49	S_0	87.3	0.94	0.91	97.0	0.87	>1800
MNIST07/01	S_3	259.7	2.10	2.06	299.6	2.03	15.1
MNIST07/02	S_2	310.1	2.54	2.59	333.1	2.59	>1800
MNIST14/01	S_3	744.0	9.56	9.60	1010.3	9.91	19.7
MNIST14/02	S_2	872.0	13.0	14.8	1243.5	14.9	>1800
MNIST28/01	S_3	>1800	97.9	107.7	>1800	56.3	36.3
MNIST28/02	S_3	>1800	232.7	232.2	>1800	179.9	52.7

Table 4.6: Running times of basic algorithms for determining linear separability.

small norm solution, and uses nearly twice as many variables to achieve this). It is interesting to see that it can pay off to apply the slower basic method in the incremental algorithm. The reason for that is that the hyperplanes found by LSEP_2 have larger margin, and therefore less of them are needed.

On the largest datasets (MNIST28/01 and MNIST28/02) LSEPY_2 was significantly faster than LSEPX_2. This demonstrates that removing points from the active sets can convey large speedups in certain cases. It can also be observed that on the largest datasets the dual based LSEPZ was the fastest method, while on the other datasets it was the slowest one.

The running times of the proposed hybrid algorithms (LSEPZX and LSEPZY) are shown in Table 4.8. The order of the methods according to the average rank is: 1. LSEPZY_2 (1.9), 2. LSEPZX_1 (2.25), 3. LSEPZX_2 (2.6), 4. LSEPZY_2 (3.25). The differences between the running times are not as drastic as in the previous experiments. This is because the first step of the algorithm is the same in each case: running LSEPZ and possibly reducing the number of features. The unique second step is run only on the reduced dataset.

If we compare primal based basic methods with the proposed ones, then it turns out that the proposed algorithms are better. Typically, the speedup factor between a basic method and its corresponding LSEPX/LSEPY/LSEPZX/LSEPZY variant is greater than 100.

The dual based basic methods (LSEP_1^*, LSEP_1^+ and LSEP_2^*) are sometimes slightly faster than the proposed LSEPX/LSEPY/LSEPZX/LSEPZY algorithms, but in contrast with the proposed methods they do not construct a separating hyperplane in the separable case.

Convex separability

The running times of basic methods for determining convex separability can be seen in Table 4.9. Each experiment consisted of two runs: at first the first class was the inner set, and then the second. The numbers shown in the table are the summed running times of the two runs (given in seconds).

Problem		Running time (seconds)				
		LSEPX$_1$	LSEPX$_2$	LSEPY$_1$	LSEPY$_2$	LSEPZ
VOTES	S_1	0.029	0.041	0.029	0.038	0.11
WISCONSIN	S_1	0.049	0.031	0.048	0.055	0.081
MNIST04/24	S_1	0.27	0.28	0.47	0.37	2.63
MNIST04/49	S_0	0.26	0.36	0.36	0.36	2.37
MNIST07/01	S_3	1.08	0.68	1.55	0.69	4.52
MNIST07/02	S_2	0.39	0.49	0.67	1.57	12.7
MNIST14/01	S_3	31.2	2.44	36.2	2.10	7.45
MNIST14/02	S_2	5.38	3.59	5.67	4.78	133.5
MNIST28/01	S_3	>1800	388.0	>1800	88.1	14.8
MNIST28/02	S_3	>1800	640.6	>1800	284.9	143.2

Table 4.7: Running times of LSEPX, LSEPY, and LSEPZ.

Problem		Running time (seconds)			
		LSEPZX$_1$	LSEPZX$_2$	LSEPZY$_1$	LSEPZY$_2$
VOTES	S_1	0.11	0.11	0.11	0.11
WISCONSIN	S_1	0.088	0.083	0.093	0.083
MNIST04/24	S_1	2.61	2.96	3.01	2.70
MNIST04/49	S_0	2.39	2.49	2.37	2.74
MNIST07/01	S_3	5.70	5.40	5.54	5.17
MNIST07/02	S_2	12.7	12.9	12.6	13.6
MNIST14/01	S_3	10.2	16.3	15.9	8.24
MNIST14/02	S_2	136.1	151.2	160.5	161.0
MNIST28/01	S_3	26.1	27.0	31.1	15.5
MNIST28/02	S_3	338.1	228.8	378.0	215.0

Table 4.8: Running times of LSEPZX and LSEPZY.

Problem		Running time (seconds)					
		CSEP_{X2}	CSEP_{Y2}	CSEP_{ZX2}	CSEP_{ZY2}	CSEP_2^*	CSEP_Z
VOTES	S_1	0.78	0.80	1.47	1.48	4.38	5.36
WISCONSIN	S_1	0.62	0.53	0.89	0.92	5.42	4.13
MNIST04/24	S_1	44.0	49.4	136.8	143.4	>1800	>1800
MNIST04/49	S_0	4.63	5.65	11.8	24.7	>1800	>1800
MNIST07/01	S_3	10.3	10.8	29.2	29.4	>1800	>1800
MNIST07/02	S_2	58.1	58.8	152.6	170.3	>1800	>1800
MNIST14/01	S_3	54.6	51.8	68.7	67.8	>1800	>1800
MNIST14/02	S_2	>1800	>1800	310.5	375.1	>1800	>1800
MNIST28/01	S_3	>1800	>1800	221.3	222.4	>1800	>1800
MNIST28/02	S_3	>1800	>1800	>1800	>1800	>1800	>1800

Table 4.9: Running times of basic algorithms for determining convex separability.

.

Problem		Pruning efficiency	Problem		Pruning efficiency
VOTES	S_1	98.92 %	MNIST07/02	S_2	98.91 %
WISCONSIN	S_1	93.24 %	MNIST14/01	S_3	99.99 %
MNIST04/24	S_1	47.62 %	MNIST14/02	S_2	100 %
MNIST04/49	S_0	12.69 %	MNIST28/01	S_3	100 %
MNIST07/01	S_3	99.73 %	MNIST28/02	S_3	100 %

Table 4.10: Percentage of outer points cut by the centroid method (CSEPC).

The order of the basic methods according to the average rank measure is: 1. CSEP_{X2} (2.15), 2. CSEP_{Y2} (2.45), 3. CSEP_{ZX2} (2.75), 4. CSEP_{ZY2} (3.45), 5–6. CSEP_2^*, CSEP_Z (5.1). The dual based methods (CSEP_2^* and CSEP_Z) performed much worse than the other ones. This is because the dual based methods are not able to cut more than 1 outer point in 1 iteration. None of the basic methods was able to finish on MNIST28/02 in less than 1800 seconds.

The proposed CSEPX approach consist of two steps: at first the outer set is pruned by running the centroid (CSEPC) method, and then a basic method (CSEP) is run on the pruned dataset. The more outer points CSEPC can cut, the smaller problem the CSEP step has to solve, therefore the "pruning efficiency" of the CSEPC step can greatly influence the running time of CSEPX.

Table 4.10 shows the percentage of outer points cut by CSEPC for each dataset. (In each cell the values of the corresponding 2 runs are averaged.) We can see that the pruning efficiency was typically high on the given problems: in 8 cases it was greater than 90 %, and in 3 cases it was 100 %.

The running times of the proposed CSEPC and CSEPX methods can be seen in Table 4.11

Problem		Running time	Problem		Running time
VOTES	S_1	0.043	MNIST07/02	S_2	1.69
WISCONSIN	S_1	0.041	MNIST14/01	S_3	0.40
MNIST04/24	S_1	11.3	MNIST14/02	S_2	2.47
MNIST04/49	S_0	15.2	MNIST28/01	S_3	0.85
MNIST07/01	S_3	0.26	MNIST28/02	S_3	6.26

Table 4.11: Running times of the centroid method (CSEPC).

Problem		Running time (seconds)					
		CSEPX$_{X2}$	CSEPX$_{Y2}$	CSEPX$_{ZX2}$	CSEPX$_{ZY2}$	CSEPX$^*_2$	CSEPX$_Z$
VOTES	S_1	0.14	0.17	0.24	0.26	0.080	0.13
WISCONSIN	S_1	0.60	0.53	0.80	0.84	0.51	0.61
MNIST04/24	S_1	53.9	51.4	111.6	116.8	>1800	>1800
MNIST04/49	S_0	22.2	23.8	50.4	44.2	>1800	>1800
MNIST07/01	S_3	3.21	3.48	11.6	12.0	20.3	13.1
MNIST07/02	S_2	21.0	20.1	58.4	58.5	125.5	62.2
MNIST14/01	S_3	0.86	0.86	1.18	1.25	1.98	1.04
MNIST14/02	S_2	2.47	2.47	2.47	2.47	2.47	2.47
MNIST28/01	S_3	0.85	0.85	0.85	0.85	0.85	0.85
MNIST28/02	S_3	6.26	6.26	6.26	6.26	6.26	6.26

Table 4.12: Running times of enhanced algorithms for determining convex separability.

and Table 4.12. The order of CSEPX variants according to the average measure rank is: 1–2. CSEPX$_{X2}$, CSEPX$_{Y2}$ (2.4), 3. CSEPX$_{ZX2}$ (3.75), 4. CSEPX$_Z$ (4.05), 5. CSEPX$^*_2$ (4.15), 6. CSEPX$_{ZY2}$ (4.25). The difference between the running times is smaller than in the case of basic methods, since each CSEPX variant runs CSEPC as the first step.

If we compare the best basic method (CSEP$_{X2}$) with the best proposed one (CSEPX$_{X2}$), then we can observe the following: In the case of the S_0-typed MNIST04/49 problem the basic method was ∼5 times faster, and in the case of the S_1-typed MNIST04/24 it was slightly faster. This situation can happen, because CSEP$_{X2}$ exits immediately after finding an inseparable outer point, while the CSEPC step of CSEPX$_{X2}$ cannot do this. In every other cases, CSEPX$_{X2}$ was faster than CSEP$_{X2}$, often with orders of magnitude. For example, in the case of S_3-typed MNIST14/01 the speedup factor was ∼60, and in the case of MNIST28/01 it was more than 2000. In terms of average and maximal running time the proposed CSEPX$_{X2}$ method is far better than CSEP$_{X2}$, indicating that the performance of the proposed method is more stable.

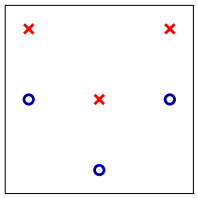
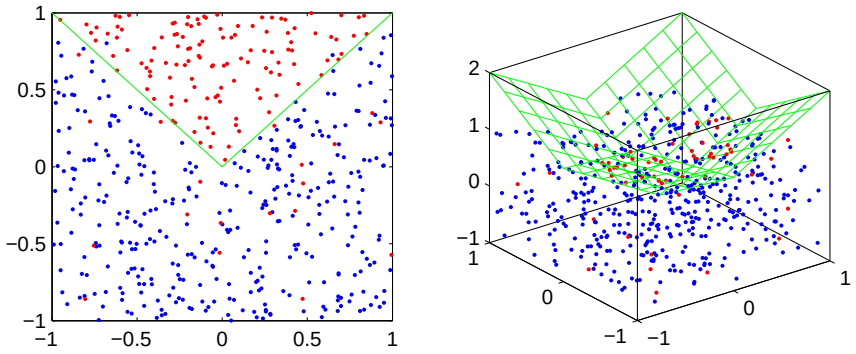


Figure 4.2: The TOY dataset.



(a) (b)

Figure 4.3: 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.

4.2 Experiments with 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 4.2). 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 4.3).

- **V3**: The 3-dimensional ($d = 3$, $n = 10^5$) version of the previous dataset (see Figure 4.3).

The real-life datasets involved in the experiments were extracted from the UCI machine learning repository [Asuncion and Newman, 2007]. 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 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**: Fisher discriminant analysis (see page 14 for the description). Regularization was done by adding the term $\lambda \mathbf{w}^T \mathbf{w}$ to the denominator of the objective function. The sole parameter of the algorithm is the regularization coefficient λ (default value: 10^{-6}).
- **LOGR**: Logistic regression (see page 14) with L2 regularization applied on the weight vector $\mathbf{w}$ (see page 22). The minimization of the objective function was done by Newton's method. The starting point was the zero vector. The algorithm has 2 parameters: the regularization coefficient λ (default value: 10^{-6}) and the number of iterations E.
- **SPER**: Smooth perceptron (see page 15) with L2 regularization applied on the weight vector $\mathbf{w}$. 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.
- **ALN**: Adaptive linear neuron (see page 16) with L2 regularization applied on the weight vector $\mathbf{w}$. The only parameter of the algorithm is the regularization coefficient λ (default value: 10^{-6}).
- **LSVM**: Linear support vector machine (see page 16). The algorithm has one parameter: the tradeoff coefficient C.
- **KNN**: K nearest neighbors (see page 16). The only parameter of the approach is the number of relevant neighbors K.
- **ID3**: ID3 decision tree (see page 17). 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 (see page 18) with L2 regularization applied on matrix $\mathbf{W}$ and vector $\mathbf{v}$. 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 with Gaussian kernel (see page 18). The only parameter of the algorithm is the tradeoff coefficient C.
- **MR**: Convex polyhedron classifier with maximal rejection based training (see page 42). 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 42). The parameters of the algorithm are the training method (G: gradient method, see page 51, N: Newton's method, see page 53, 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, 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 [Chang and Lin, 2001] implementation was used, via the built-in Python interface. All other algorithms were implemented from scratch in Python [Rossum, 2006], using the NumPy [Ascher et al., 2001] 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.

4.2.1 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 4.13. 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:

variant	loss	h	smax	method	parameters	$\|\mathbf{W}\|$	$\mathcal{L}_{01}$	$\mathcal{L}$
#1	S	A	A1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	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	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	9.6	0	0.0474
#4	S	A	B1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	9.6	0	0.0487
#5	S	A	B1	N	$E_2 = 10$, $R = 1$	9.6	0	0.0483
#6	S	A	B1	G+N	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	9.6	0	0.0483
#7	S	A	C1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	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	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	9.3	0	0.0476
#10	S	B	A1	G	$E = 1000$, $\eta = 0.001$, $R = 1$	2.4	0	0.0013
#11	S	B	A1	N	$E_2 = 10$, $R = 1$	2.4	0	0.0014
#12	S	B	A1	G+N	$E = 200$, $\eta = 0.001$, $E_2 = 10$, $R = 1$	2.4	0	0.0013
#13	S	B	B1	G	$E = 1000$, $\eta = 0.001$, $R = 1$	2.4	0	0.0015
#14	S	B	B1	N	$E_2 = 10$, $R = 1$	2.4	0	0.0015
#15	S	B	B1	G+N	$E = 100$, $\eta = 0.001$, $E_2 = 10$, $R = 1$	2.4	0	0.0014
#16	S	B	C1	G	$E = 1000$, $\eta = 0.001$, $R = 1$	2.0	0	0.0011
#17	S	B	C1	N	$E_2 = 10$, $R = 1$	0.7	1	0.3533
#18	S	B	C1	G+N	$E = 200$, $\eta = 0.001$, $E_2 = 10$, $R = 1$	2.0	0	0.0011
#19	S	C	A1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	10.2	0	0.2362
#20	S	C	A1	N	$E_2 = 20$, $R = 1$	10.2	0	0.2362
#21	S	C	A1	G+N	$E = 200$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	10.2	0	0.2362
#22	S	C	A2	G	$E = 1000$, $\eta = 0.1$, $R = 1$	9.5	0	0.0488
#23	S	C	A2	N	$E_2 = 20$, $R = 1$	9.5	0	0.0488
#24	S	C	A2	G+N	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	9.5	0	0.0488
#25	S	C	B1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	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	$E = 100$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	10.5	0	0.1442
#28	S	C	B2	G	$E = 1000$, $\eta = 0.1$, $R = 1$	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	$E = 200$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	8.8	0	0.1623
#31	S	C	C1	G	$E = 1000$, $\eta = 0.1$, $R = 1$	10.0	0	0.0714
#32	S	C	C1	N	$E_2 = 10$, $R = 1$	4.0	1	0.3665
#33	S	C	C1	G+N	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	10.0	0	0.0714
#34	S	C	C2	G	$E = 1000$, $\eta = 0.1$, $R = 1$	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	$E = 50$, $\eta = 0.1$, $E_2 = 10$, $R = 1$	9.3	0	0.0464
#37	L	A	A1	G	$E = 1000$, $\eta = 0.05$, $R = 1$	17.5	0	0.1652
#38	L	A	A1	N	$E_2 = 10$, $R = 1$	17.5	0	0.1620
#39	L	A	A1	G+N	$E = 200$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.5	0	0.1620
#40	L	A	B1	G	$E = 1000$, $\eta = 0.05$, $R = 1$	17.5	0	0.1662
#41	L	A	B1	N	$E_2 = 10$, $R = 1$	17.5	0	0.1592
#42	L	A	B1	G+N	$E = 50$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.5	0	0.1639
#43	L	A	C1	G	$E = 1000$, $\eta = 0.05$, $R = 1$	17.2	0	0.1643
#44	L	A	C1	N	$E_2 = 10$, $R = 1$	17.2	0	0.1508
#45	L	A	C1	G+N	$E = 50$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.2	0	0.1638
#46	L	C	B1	G	$E = 1000$, $\eta = 0.05$, $R = 1$	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	$E = 50$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.1	0	0.8210
#49	L	C	B2	G	$E = 1000$, $\eta = 0.05$, $R = 1$	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	$E = 100$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	14.8	0	0.8783
#52	L	C	C1	G	$E = 1000$, $\eta = 0.05$, $R = 1$	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	$E = 50$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.1	0	0.4094
#55	L	C	C2	G	$E = 1000$, $\eta = 0.05$, $R = 1$	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	$E = 50$, $\eta = 0.05$, $E_2 = 10$, $R = 1$	17.3	0	0.1582

Table 4.13: Results of SMAX training on the TOY dataset.

- **SA** (loss = 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 smax = smax_{C1}.
- **SB** (loss = 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 smax = smax_{C2}.
- **SC** (loss = 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 smax = smax_{C2}.
- **LA** (loss = 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 lowest $\mathcal{L}$ value was achieved by smax = smax_{C1}.
- **LC** (loss = 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 smax = smax_{C2}.

4.2.2 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 [Egan, 1975]. 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 $\text{TP}_0 = 0$, $\text{FP}_0 = 0$, $\text{TP}_k = \sum_{i=1}^{k} y_{(i)} / \sum_{i=1}^{m} y_i$, and $\text{FP}_k = \sum_{i=k+1}^{m} y_{(i)} / \sum_{i=1}^{m} (1 - y_i)$ $(k = 1 \ldots, m)$, then the AUC value can be written as:

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

  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.

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 4.14: Results of classification algorithms on the V2 dataset.

The results of classification algorithms on the V2 dataset can be seen in Table 4.14. 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 ΔAUC/VTIME, it was the best one.

Results on the V3 dataset can be seen in Table 4.15. 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 ΔAUC/VTIME, it was the best one.

Results for the ABALONE dataset can be seen in Table 4.16. 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 ΔAUC/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 4.17. 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 ΔAUC/VTIME, it was the second best one (beaten by MLP, tied with LOGR).

Results for the CHESS dataset can be seen in Table 4.18. 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, 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

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 4.15: 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 4.16: 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 4.17: Results of classification algorithms on the BLOOD dataset.

of AUC, and it was the best method in terms of ΔAUC/VTIME.

Results for the SEGMENT dataset are shown in Table 4.19. 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 ΔAUC/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 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).

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++.

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 4.18: Results of classification algorithms on the CHESS dataset.

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 4.19: Results of classification algorithms on the SEGMENT dataset.

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".

Notes on setting meta-parameters

Many of the algorithms involved in the experiments have meta-parameters that should be set appropriately in order to get a reasonable performance. For doing this, I generally applied the following heuristic greedy search method:

1. Choose a meta-parameter to optimize based on intuition, or draw it randomly, if we have no guess!
2. Optimize the value of the selected meta-parameter by "doubling and halving" (if it is numerical) or by exhaustive search (if it is categorical)!
3. Go to step 1, if we want to continue optimizing!

It is important to note that meta-parameter optimization should use a different dataset or at least a different cross-validation split as the main experiment (in this work the latter solution was applied).

One may notice that the proposed SMAX approach has more meta-parameters than the other algorithms involved in the comparison. This is true, but I have found that for the given test problems the performance is not very sensitive to the values of most meta-parameters. The meta-parameters with largest influence on performance are the number of hyperplanes K, the range of random initialization R, the learning rate η, and the number of epochs E.

4.3 Experiments with collaborative filtering

The NETFLIX dataset

This collaborative filtering dataset is currently 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 [Bennett and Lanning, 2007].

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 (see also Figure 4.4).

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.

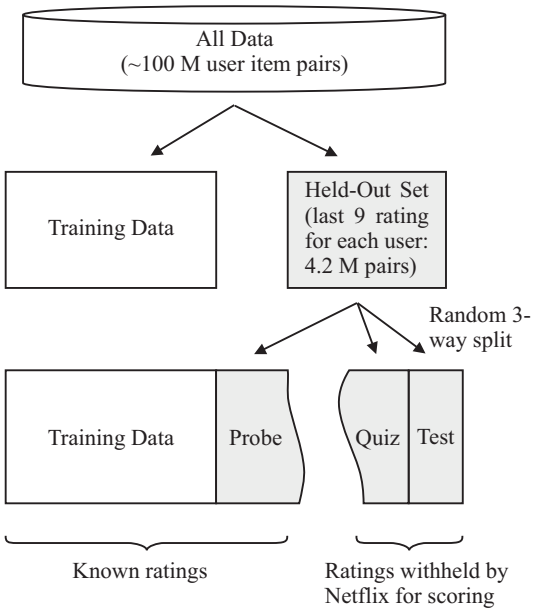


Figure 4.4: The train–test split and the naming convention of the NETFLIX dataset (after [Bell et al., 2007])

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.[2] 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.[3] Note that the RMSE on the Test set is withheld by Netflix.

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

[2]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.

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

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 [Bell et al., 2007].

- 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.

Experiments

The algorithms involved in the experiments were the following:

- **DC**: Double centering (see page 24 for the details). The only parameter of the algorithm is the number of epochs E (default value: 2).
- **BRISMF**: Biased regularized incremental simultaneous matrix factorization (see page 25). 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 (see page 27). 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).
- **SMAX$_{\mathrm{CF}}$**: The proposed smooth maximum based convex polyhedron approach (see page 54). 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 4.20. Recall that SMAX$_{\mathrm{CF}}$ can be considered as a generalization of BRISMF. We can see, that the SMAX$_{\mathrm{CF}}$ 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$_{\mathrm{CF}}$, and not surprisingly, DC was the worst in terms of RMSE. It is true for all of BRISMF, NSVD1, and SMAX$_{\mathrm{CF}}$ 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

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 4.20: Results of collaborative filtering algorithms on the NETFLIX dataset.

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 (LINR, see page 21). The value of the regularization coefficient was $\lambda = 1.4$. The results can be seen in Table 4.21.

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 LINR 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.	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 4.21: Results of linear blending on the NETFLIX dataset.

Conclusion

This thesis was about convex polyhedron based methods for machine learning. The first chapter (Introduction) briefly introduced the field of machine learning and located convex polyhedron learning in it. The second chapter (Algorithms) dealt with the problem of linear and convex separation, and gave algorithms for training convex polyhedron classifiers and predictors. The third chapter (Model complexity) collected known facts about the Vapnik–Chervonenkis dimension of convex polyhedron classifiers and proved new results. The fourth chapter (Applications) presented experiments with the given algorithms on real and artificial datasets.

The summary of new scientific results contained in the thesis is the following:

- Direct and incremental approaches were investigated for determining the linear separability of point sets. Heuristics were proposed for choosing the active constraints and variables of the next step (LSEPX, LSEPY, LSEPZX, LSEPZY). A novel algorithm with low time requirement was given for the approximate convex separation of point sets (CSEPC). A novel exact algorithm with low expected time requirement was introduced for determining the convex separability of point sets (CSEPX).

- The possibility of approximating the maximum operator by smooth functions was investigated, and six, parameterizable smooth maximum function families were introduced. A novel, smooth maximum function based approach was introduced for training convex polyhedron classifiers (SMAX). A novel, smooth maximum function based algorithm was given for training convex polyhedron models in the case of collaborative filtering ($\mathrm{SMAX}_{\mathrm{CF}}$).

- The Vapnik–Chervonenkis dimension of 2-dimensional convex K-gon classifiers was determined so that the label of the inner (convex) region is unrestricted. A new lower bound was proved for the Vapnik–Chervonenkis dimension of d-dimensional convex K-polyhedron classifiers. In the special cases $d = 3$ and $d = 4$ the bound was further improved.

- Scalable and accurate algorithms were introduced for collaborative filtering. A novel matrix factorization technique called BRISMF (biased regularized incremental simultaneous matrix factorization) was introduced. A new training algorithm for Paterek's NSVD1 model was given.

List of publications

[P1] 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.

[P2] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix factorization and neighbor based algorithms for the Netflix Prize problem. *Proc. of the 2008 ACM Conference on Recommender Systems (RECSYS'08)*, pages 267–274, Lausanne, Switzerland, 2008.

[P3] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Investigation of various matrix factorization methods for large recommender systems. *Proc. of the 2nd KDD Workshop on Large Scale Recommender Systems and the Netflix Prize Competition*, Las Vegas, Nevada, USA, 2008.

[P4] G. Takács, I. Pilászy, B. Németh, and D. Tikk. A unified approach of factor models and neighbor based methods for large recommender systems. *Proc. of the 1th IEEE ICADIWT Workshop on Recommender Systems and Personalized Retrieval*, pages 186–191, Ostrava, Czech Republic, 2008.

[P5] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the Gravity Recommendation System. *ACM SIGKDD Explorations Newsletter*, 9(2): 80–83, 2007.

[P6] G. Takács, I. Pilászy, B. Németh, and D. Tikk. On the Gravity Recommendation System. *Proc. of the KDD Cup and Workshop 2007*, pages. 22–30, San Jose, California, USA, 2007.

[P7] G. Takács and B. Pataki. Case-level detection of mammographic masses. *International Journal of Applied Electromagnetics and Mechanics*, 25(1–4): 395–400, 2007.

[P8] G. Takács. The Vapnik–Chervonenkis dimension of convex n-gon classifiers. *Hungarian Electronic Journal of Sciences*, 2007.

[P9] G. Takács and B. Pataki. Lower bounds on the Vapnik–Chervonenkis dimension of convex polytope classifiers. *Proc. of the 11th International Conference on Intelligent Engineering Systems (INES 2007)*, Budapest, Hungary, 2007.

[P10] G. Takács and B. Pataki. Deciding the convex separability of pattern sets. *Proc. of the 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'2007)*, Dortmund, Germany, 2007.

[P11] G. Takács and B. Pataki. An efficient algorithm for deciding the convex separability of point sets. *Proc. of the 14th PhD Mini-Symposium, Budapest University of Technology and Economics, Department of Measurement and Information Systems*, pages 54–57, Budapest, Hungary, 2007.

[P12] G. Takács and B. Pataki. Nearest local hyperplane rules for pattern classification. *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 302–313, Rome, Italy, 2007.

[P13] G. Takács and B. Pataki. A lépcsőzetes döntéshozás elvének műszaki alkalmazásai, (in Hungarian). *Elektronet*, 16(8): 76–78, 2007.

[P14] M. Altrichter, G. Horváth, B. Pataki, Gy. Strausz, G. Takács and J. Valyon. *Neurális hálózatok*, (in Hungarian). Panem, 2006.

[P15] G. Takács and B. Pataki. Local hyperplane classifiers. *Proc. of the 13th PhD Mini-Symposium, Budapest University of Technology and Economics, Department of Measurement and Information Systems*, pages 44–45, Budapest, Hungary, 2006.

[P16] G. Takács and B. Pataki. Fast detection of masses in mammograms with difficult case exclusion. *International Scientific Journal of Computing*, 4(3): 70–75, 2005.

[P17] G. Takács and B. Pataki. Case-level detection of mammographic masses. *Proc. of the 12th International Symposium on Interdisciplinary Electromagnetic, Mechanic and Biomedical Problems (ISEM 2005)*, pages 214–215, Bad Gastein, Austria, 2005.

[P18] G. Takács and B. Pataki. Fast detection of mammographic masses with difficult case exclusion. *Proc. of the 3rd IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'2005)*, pages 424–428, Sofia, Bulgaria, 2005.

[P19] G. Takács and B. Pataki. Computer-aided detection of mammographic masses. *Proc. of the 12th PhD Mini-Symposium, Budapest University of Technology and Economics, Department of Measurement and Information Systems*, pages 24–25, Budapest, Hungary, 2005.

[P20] N. Tóth, G. Takács, and B. Pataki. Mass detection in mammograms combining two methods. *Proc. of the 3rd European Medical & Biological Engineering Conference (EMBEC'05)*, Prague, Czech Republic, 2005.

[P21] G. Horváth, B. Pataki, Á. Horváth, G. Takács, and G. Balogh. Detection of microcalcification clusters in screening mammography. *Proc. of the 3rd European Medical & Biological Engineering Conference (EMBEC'05)*, Prague, Czech Republic, 2005.

[P22] G. Takács. The smooth maximum classifier. Accepted at: Second Győr Symposium on Computational Intelligence, 2009.

[P23] G. Takács. Smooth maximum based algorithms for classification, regression, and collaborative filtering. Accepted at: *Acta Technica Jaurinensis, Series Intelligentia Computatorica*, 2009.

[P24] G. Takács. Efficient algorithms for determining the linear and convex separability of point sets. Accepted at: *Acta Technica Jaurinensis, Series Intelligentia Computatorica*, 2009.

[P25] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Unifying collaborative filtering approaches. Veszprém Optimization Conference: Advanced Algorithms (VOCAL 2008), Veszprém, Hungary, 2008.

[P26] R. Horváth-Bokor, Z. Horváth, and G. Takács. Kockázatelemzés logisztikus regresszióval nagy adathalmazokon, (in Hungarian). 28. Magyar Operációkutatási Konferencia, Balatonőszöd, Hungary, 2009.

Bibliography

K. Appel and W. Haken. Every planar map is four colorable. *Ilinois Journal of Mathematics*, 21:439–567, 1977.

E.M. Arkin, F. Hurtado, J.S.B. Mitchell, C. Seara, and S.S. Skiena. Some lower bounds on geometric separability problems. *International Journal of Computational Geometry and Applications*, 161:1–26, 2006.

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

P. Assouad. Densité et dimension. *Annales de l'Institut Fourier*, 33:233–282, 1983.

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

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

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.

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

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.

O. Bousquet, S. Boucheron, and G. Lugosi. Introduction to statistical learning theory. *Lecture Notes in Artificial Intelligence*, 3176:169–207, 2004.

P.S Bradley, U.M. Fayyad, and O.L. Mangasarian. Mathematical programming for data mining: Formulations and challenges. *INFORMS Journal on Computing*, 11(3):217–238, 1999.

L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993.

C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

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/`.

V. Chvátal. *Linear programming*. W. H. Freeman & Co., 1983.

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

L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic theory of pattern recognition*. Springer, New York, 1996.

D.P. Dobkin and D. Gunopulos. Concept learning with geometric hypotheses. In *Proc. 8th Annual Conference on Computational Learning Theory*, pages 329–336. ACM Press, New York, 1995.

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

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.

T.S. Ferguson. Linear programming – A concise introduction, 2004.
URL: `http://www.math.ucla.edu/~tom/LP.pdf`.

T. Finley. PyGLPK, version 0.3, 2008.
URL: `http://www.cs.cornell.edu/~tomf/pyglpk/`.

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.

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

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

S. Funk. Netflix update: Try this at home, 2006.
URL: `http://sifter.org/~simon/journal/20061211.html`.

L. Györfi, M. Kohler, A. Krzyzak, and H. Walk. *A distribution-free theory of nonparametric regression*. Springer, New York, 2002.

D. Haussler and E. Welzl. Epsilon nets and simplex range queries. *Discrete Computational Geometry*, 2:127–151, 1987.

S. Haykin. *Neural networks and learning machines*. Prentice Hall, 3rd edition, 2008.

R. Hooke and T.A. Jeeves. "direct search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, 1961.

T. Joachims. Making large-scale SVM learning practical. In B. Schlkopf, C. Burges, and A. Smola, editors, *Advances in kernel methods - Support vector learning*. MIT Press, 1999.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for python, 2001.
URL: `http://www.scipy.org/`.

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4: 373–395, 1984.

J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(1):502–506, 1953.

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.

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

Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1999.
URL: `http://yann.lecun.com/exdb/mnist/`.

A. Makhorin. GNU Linear Programming Kit, version 4.37, 2009.
URL: `http://www.gnu.org/software/glpk/`.

W. McCullogh and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Biophysics*, 7:115–133, 1943.

N. Megiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, 3:325–337, 1988.

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

K. Pearson. *The life, letters and labors of Francis Galton*. Cambridge University Press, 1930.

K. Pearson. Mathematical contributions to the theory of evolution. III. Regression, heredity and panmixia. *Philosophical Transactions of the Royal Society of London*, 187:253–318, 1896.

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.

J.C. Platt. *Fast training of support vector machines using sequential minimal optimization*, pages 185–208. MIT Press, 1999.

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

C.C. Rodríguez. Learning theory notes, VC dimension: Examples and tools, 2004. URL: `http://omega.albany.edu:8008/ml/`.

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

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

N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory (A)*, 13:145–147, 1972.

J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Carnegie Mellon University, School of Computer Science, 1994.

C. Stone. Consistent nonparametric regression. *Annals of Statistics*, 8:1348–1360, 1977.

G. Takács. The Vapnik–Chervonenkis dimension of convex n-gon classifiers. *Hungarian Electronic Journal of Sciences*, 2007.

G. Takács and B. Pataki. Deciding the convex separability of pattern sets. In *Proc. of the 4th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS'2007)*, pages 278–80, 2007a.

G. Takács and B. Pataki. Lower bounds on the Vapnik–Chervonenkis dimension of convex polytope classifiers. In *Proc. of the 11th International Conference on Intelligent Engineering Systems (INES 2007)*, 2007b.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. On the Gravity Recommendation System. In *Proc. of the KDD Cup and Workshop 2007 (KDD 2007)*, pages 22–30, 2007.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. A unified approach of factor models and neighbor based methods for large recommender systems. In *Proc. of the 1st IEEE ICADIWT Workshop on Recommender Systems and Personalized Retrieval*, pages 186–191, 2008.

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.

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

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

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.

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

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

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.