Trainable monotone combiner

Sergey Grosman
Konstanz, Germany
srg.grosman@gmail.com

May 14, 2020

Abstract

We consider a binary classification problem in which the class label is given in the form of a discriminant function that satisfies a monotone constraint. That is, the degree of confidence that an object belongs to a class can not decrease as one of the input features increases. This manuscript examines how such a discriminant function can be trained on the basis of a labeled data set. Two alternative quality measures are considered. One of them is the AUC, which is based on the ROC analysis. The second is encouraged by the Neyman-Pearson lemma, which aims to maximize the ratio of correctly classified to misclassified examples. We propose an approach in which feature space is partitioned into quality layers that can then effectively compute the discriminant function. We prove that the resulting discriminant function is optimal with respect to the two quality measures mentioned, which indicates, among other things, the equivalence of these two quality measures. We also show that the associated optimal partitioning of feature space is unique, and provide a polynomial training algorithm for generating this partitioning.

1 Introduction

Classification tasks, as part of the field of machine learning, belong to the fastest growing research topics of recent decades. Simply put, to classify means to map a point in the feature space to one of the possible classes. Within machine learning, the classification task is to train this mapping using labeled examples, that is, based on the points in the feature space at which the corresponding class is known. Given an arbitrary point in the feature space, the trained mapping selects a concrete one of the possible classes. Another possibility is to assign a degree of belief or a *score* to each class, which can be interpreted as a function applied to a feature vector x. This function is called a discriminant function - the greater its value $score(x)$ for a certain class, the more likely it is that x belongs to that class. This approach gives a user greater flexibility than choosing a fixed class. In this work, we consider a binary, that is, a two-class classification problem in which, for the sake of simplicity, we denote the two classes *positives*

and *negatives*. We let $score(x)$ be the value of the discriminant function for the class *positives*, then $1 - score(x)$ is the value for the class *negatives*. The larger the value of $score(x)$, the greater the probability that x belongs to the class *positives*.

There are a number of applications in which the discriminant function is expected to be monotonically non-decreasing with respect to each of the input features. For example, a creditworthiness of a borrower is not expected to decrease as her or his income increases. For such applications, the monotone constraints are included in the classification model, and one speaks of a monotone classification [1]. In addition to the above-mentioned area of credit scoring [2], such classifiers are used, for example, in the fields of medicine [3, 4] and finance [5]. The monograph [6] provides a comprehensive review of the state of the art of the monotone classification, including techniques, algorithms, quality measures, and data sets. These techniques aim to associate a class of the ordered class set with an input feature vector so that the associated class can not decrease with the increasing feature values. In Section 9 we will compare the existing approaches in detail with the approach of this paper, and also outline the possibilities to extend our approach to multi-class set of ordered classes.

There is another use case of monotone classification – this is the task of combining classifiers of a general nature, also called base classifiers. In this case, the input feature vector for a monotone classifier consists of the discriminant function values of the base classifiers, and we interchangeably call our monotone classifier a monotone combiner, bearing in mind that it can be used both to combine general features and to combine the output of base classifiers. Please refer to [7, 8] for a comprehensive overview of techniques for creating ensembles of base classifiers and possible classifier combination methods. Following the notation of [7] we refer to the space containing results of the base classifiers as the *intermediate feature space*.

In this work we restrict ourselves to combiners that are monotone. To illustrate that this restriction indeed makes sense, let us look at the following example. Suppose we have two base classifiers and two input objects to classify. For the first object, both classifiers deliver 0.5 as the degree of support, and for the second object both 0.6, so that the two corresponding vectors in the intermediate feature space are $x_1 = (0.5, 0.5)^\top$ and $x_2 = (0.6, 0.6)^\top$. The following question arises:

Should we ever allow the combiner to value x_1 better than x_2?
Or, in terms of discriminant function, should we ever accept
$score(x_1) > score(x_2)$*?*

If we do not consider additional analysis of the original feature space of base classifiers, the obvious answer would be no. In fact, the better the output of the base classifiers, the higher the combined output can be expected. In other words, we can expect that the combiner of classifiers, as a mapping from the intermediate feature space to a single score, will be monotonically increasing. If a combiner ever tells to value $(0.6, 0.6)^\top$ lower than $(0.5, 0.5)^\top$, this may indicate

either an overfitting, or a suspiciously unlucky correlation of base classifiers. In both cases, it is worthwhile to revise the combiner and the base classifiers.

In this work, we begin with monotonicity, which we define as an explicit constraint, and tackle the task of feature fusion. We build a monotone combiner by partitioning the feature space into layers so that:

- Points within one layer have the same score value.
- Each layer is either strictly above or strictly below every other layer, and accordingly gets a larger or smaller score assigned.

When applying the combiner, it is important to determine within which layer a given point lies. The point then inherits the score value of the corresponding layer. In this work, we show how for a given partitioning and a point, the layer containing this point can be effectively found.

Another important topic that we pay most attention to is the meaningful generation of the described partitioning. This partitioning must be built on the basis of an appropriate quality measure and not least in polynomial time with respect to the size of the training data set, so that the method can be practically applied. Two quality measures are considered: a measure based on the Neyman-Pearson ratio [9] that corresponds to maximizing the ratio of correctly classified to misclassified examples. The second is the AUC, which is based on the ROC analysis [10]. We show that there is a unique partitioning of feature space for each labeled data set that is optimal in terms of the first quality measure. Moreover, we show that the same partitioning is optimal in terms of the second quality measure as well. Among others, this means that Neyman-Pearson optimality and AUC optimality are equivalent for monotone combiners. Finally, we propose a training algorithm to construct this optimal partitioning and thus the associated combiner in polynomial time. We call this combiner the *trainable monotone combiner*.

It should be noted that in addition to the optimality, there are two properties of great practical importance that characterize our approach:

(A) Explainability or ability to backtrack outcome. Given a result of a classifier we want to understand why the classifier has decided this way and not otherwise. Although such an analysis dramatically increases the practical acceptability of a classifier, not all classifiers allow this analysis. For neural networks, for example, explainability is practically out of the question.

Backtracking a classifier result makes sense, especially in the case of misclassification. The following two types of misclassification can be distinguished.

1. *False positive*: x falsely gets a class label c. In this case one wonders which representative of the class c within the training sample led to this labeling of x.
2. *False negative*: x would need the label c, but gets another label. As a rule, it is important to determine the next points in the feature space with the label c to see what needs to be changed so that the combination gets the label c.

Our approach offers the opportunity to backtrack both types of misclassification, as we will demonstrate in the main text.

(B) Resource conservation. The calculation of input features is often very resource intensive. It therefore makes sense to avoid the calculation of further features as far as possible, as long as the achievement of a predetermined threshold by already calculated features is guaranteed or the threshold can definitely not be reached. While we believe that the proposed monotone combiner enables such a resource-efficient implementation, an in-depth analysis of this issue would go beyond the scope of this manuscript. We formulate this issue as an open problem for future work in Section 10.

The paper is organized as follows: after this introduction, we show in Section 2 how a monotone combination of features can be derived from elementary considerations, where we also give its formal definition and some examples. In Section 3, we start with discrete monotone combiners that are used in our approach. In Section 4, we propose a way to partition the feature space using the Neyman-Pearson ratios and prove a discrete analog to the Neyman-Pearson lemma. It guarantees the optimality of the associated combiner within the family of monotone combiners. In addition, in Section 5 we show that the optimal partitioning is unique. In Sections 6 and 7 we provide a polynomial algorithm for creating this partitioning. In Section 6, we also show the AUC optimality of the associated combiner. We verify our theoretical considerations using a toy example in Section 8. We conclude by discussing the relationship of our approach to the general case of monotonic multi-class classification in Section 9 and a list of open problems in Section 10.

2 From naive thresholding to a monotone combiner

In this section we explain how a monotone combiner arises from very simple practical considerations. Recall that we are working on a binary classification problem with two classes - *positives* and *negatives*. We assume that each feature can be interpreted as a score, which means that the greater the value of the feature, the more likely it is that the object belongs to *positives*, and the less likely it is that the object belongs to *negatives*. For k features we get a k-dimensional feature vector $\bar{f} = (f_1, f_2, \ldots, f_k)^\top \in \mathbb{R}^k$. Within this semantic practical applications prompt us towards the following considerations.

Step 1: Simple thresholding. As we start trying to use multiple features together, each of which can be interpreted as a score, the first idea which comes to mind is to prescribe an individual threshold value to each of the features. The class label is then determined by checking all the inequalities of the form $f_i \geq t_i$, where t_i is the threshold value corresponding to the i-th feature. In other words the class label is determined via

$$\text{"}class\,label\text{"}(f) = \begin{cases} positive & \text{if } \bar{f} \geq \bar{t}, \\ negative & \text{otherwise.} \end{cases}$$

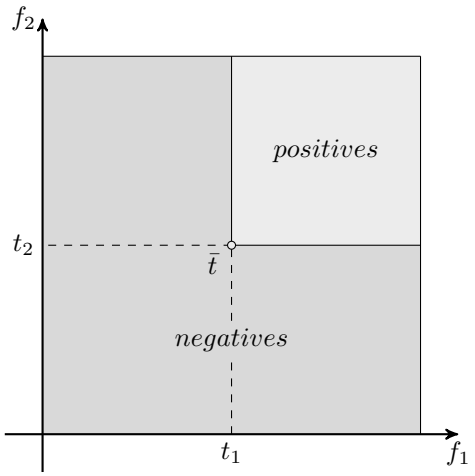


Figure 1: Thresholds combination.

We can interpret $\bar{t}$ as a vector of minimal acceptable feature values. Figure 1 illustrates the class label assignment for the dimension $k = 2$.

Step 2: Threshold refinement. In practical application one very quickly comes to the point where simple thresholding is not flexible enough. Therefore, it has to be extended to multiple vectors $\bar{t}_j$ of minimal feature values and we prescribe a *positive* class label as soon as a combination of the features values is not less than at least one of $\bar{t}_j$. The class label is then determined by

$$\text{"}class\,label\text{"}(f) = \begin{cases} positive & \text{if } \bar{f} \geq \bar{t}_j, \text{ for some } j, \\ negative & \text{otherwise.} \end{cases}$$

The collection of $\bar{t}_j$ naturally describes a better boundary between *positives* and *negatives* as can be seen in Figure 2.

Step 3: Quality levels as a discriminant function. Depending on the application and its performance measures, one may be interested in defining quality areas for the class label determination. If there is only one boundary, then the case deteriorates to the one described in Step 2. In case of multiple areas, their boundaries partition the space into stripes corresponding to the different quality levels, or in other words degrees of certainty; see Figure 3. We map each quality level to a real number (a score) $Q_i \in [0, 1]$ in such a way that the higher the degree of certainty the larger the prescribed value is. The described mapping represents nothing but a discriminant function. Due to its construction the discriminant function yields a special property, which we would like to emphasize: it is monotonically increasing with respect to the feature values. This motivates a formal definition of a monotone combination.

Definition 2.1 (Monotone combination of features)**.** As before assume a two class classification problem. Let $\bar{f} \in \mathbb{R}^k$ be a vector of feature values. We define

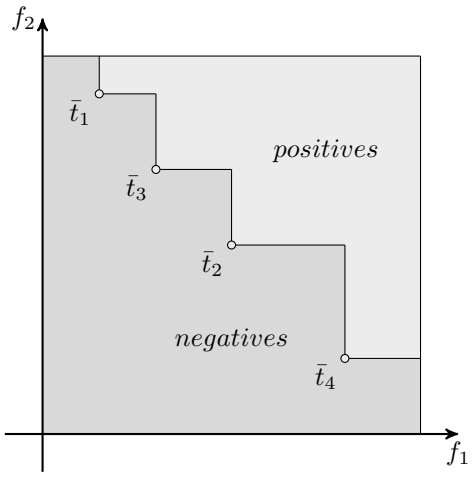


Figure 2: Complex threshold refinement.

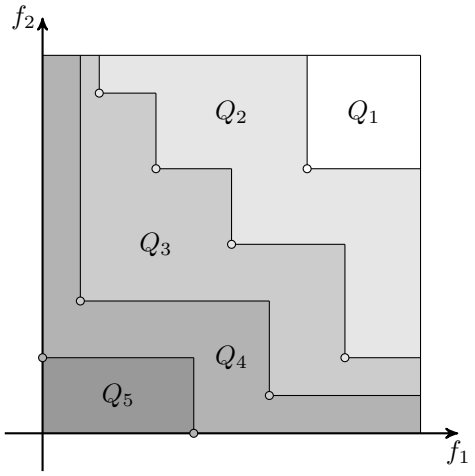
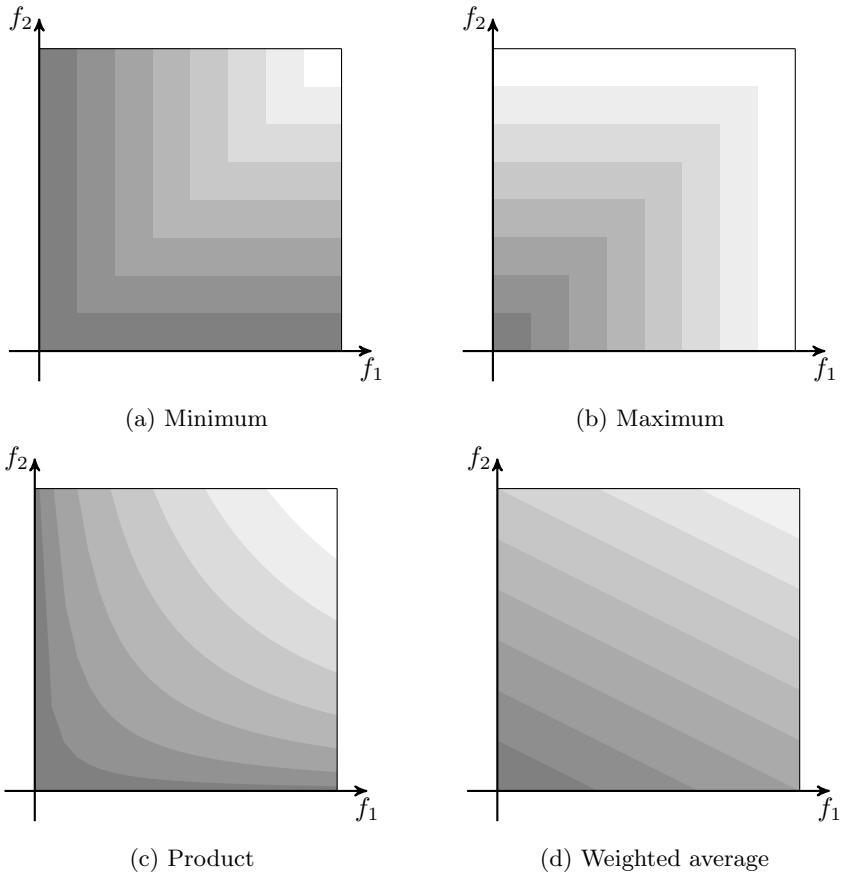


Figure 3: Quality levels $Q_5 < Q_4 < Q_3 < Q_2 < Q_1$ can be interpreted as values of the corresponding discriminant function.



(a) Minimum (b) Maximum

(c) Product (d) Weighted average

Figure 4: Isolines of different monotone combiners.

a monotone combination via its discriminant function $F : \mathbb{R}^k \to [0, 1]$ acting on the feature space, such that $F(\bar{f})$ is monotonically increasing with respect to each of the features f_i, $i = 1, \ldots, k$. It should be noted that once you have set a threshold, you will receive a class label assignment.

To monotone combiners belongs, for example, any member of the family of the so-called fixed combiners [11, 12, 13, 14], such as the product, minimum and maximum; see Figures 4a, 4b and 4c correspondingly. One further example is a weighted average [15, 16]; see Figure 4d. These and other examples can be found in [7], Chapter 5.2.

3 Discrete monotone combiner

In practical application we usually need more flexibility from the combiner than e.g. an average. As in the case for the average we are able to describe some

monotone combiners algebraically. Although we could try to construct more complex algebraic functions reflecting desired behaviour, we would like to propose another approach, inspired by the thresholding we described in Section 2.

In what follows we assume that scores are consistently prescribed for some points in the feature space and show how a monotone combiner naturally arises based on these scores. We discuss the general case first, then we are going to address clusters of points sharing the same quality level. Although the second case is the special case of the first we would like to emphasise it, as it plays a crucial role in the subsequent sections.

Let us start with the first case. Suppose a finite sample of feature vectors S is given. Unlike set of points, we allow a sample to have one entry multiple times. Assume for now that monotone scores are given on S. We address the issue of setting those scores later in the next sections. The following theorem provides a prolongation of a monotone combiner defined on a sample to the whole feature space.

Theorem 3.1. *Let $S \subset \mathbb{R}^k$ be a sample. Suppose that a discrete score function is provided $q : S \to \mathbb{R}^+$ that satisfies a monotone property:*

$$q(s_1) \geq q(s_2) \text{ as soon as } s_1 \geq s_2, \tag{1}$$

where $s_1, s_2 \in S$. Then the sample S equipped with the score function q induce a monotone combiner defined by its discriminant function:

$$score(x) = q(s(x)), \tag{2}$$

where the function $s : \mathbb{R}^k \to S$ is defined by

$$s(x) = \begin{cases} s_i \in S : s_i \leq x \text{ and } q(s_i) = \max\limits_{\substack{s_j \in S \\ s_j \leq x}} q(s_j) & \text{if } \exists s_i \in S \text{ s.t. } s_i \leq x, \\ 0 & \text{if } \nexists s_i \in S \text{ s.t. } s_i \leq x. \end{cases} \tag{3}$$

We assume $q([0, \ldots, 0]^\top) = 0$ if it is not provided explicitly.

Proof. The proof is obvious since (3) represents the searching of the point in the sample, which is not greater than a given point on the one hand and has the largest possible score on the other hand. □

Suppose that a sample S and a discrete score function $q : S \to \mathbb{R}^+$ are provided. Directly utilizing Theorem 3.1 in order to apply the combiner does not seem to be feasible since this would mean a linear searching time on the sample size $O(|S|)$. What we would propose is to group points of the sample into clusters by means of the scores. This results in boundaries of the form as in Section 2, Figure 3. It is a sophisticated approach, since we usually need a limited amount of quality levels in a practical situation. In order to evaluate a point in the feature space we have to find two sequential boundaries, which have this point in between. The search of these two boundaries can be implemented efficiently, so that its complexity depends logarithmically on the total number

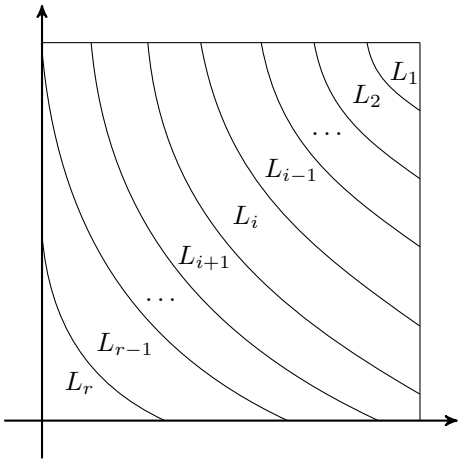


Figure 5: A layer partitioning $D = \{L_1, L_2, \ldots, L_r\}$ of the sample S.

of boundaries. Checks on whether the point is above a boundary allow efficient algorithms based on multidimensional data indices like R-trees. There is a broad range of literature on this topic; we would refer to [17, 18, 19, 20, 21].

At this point we would like to introduce a notion of layer partitioning. Each layer plays a similar role to a point in a sample. We are going to use them extensively in the subsequent sections. We start with definitions of a monotone subsample and a layer partitioning.

Definition 3.2 (Monotone subsample)**.** Let S be a sample in the feature space. We say that its subsample $T \subset S$ is a monotone subsample if $\forall t \in T \quad \forall s \in S : s \notin T$ either $s \leq t$ or s and t are not comparable with respect to the relation $\leq$. In other words $\forall t \in T \nexists s \in S : s \notin T$ and $s \geq t$.

Definition 3.3 (Layer partitioning)**.** Let S be a sample in the feature space. We say that a sequence of subsamples $\{L_1, L_2, ..., L_r\}$ is a monotone layer partitioning of S or simply a layer partitioning if $\forall i \leq r$ the union $L_1 \cup L_2 \cup ... \cup L_i$ is a monotone subsample of S. We also call L_i-s layers of the corresponding partitioning or simply layers (see Figure 5).

Following the lines of Theorem 3.1 we get

Theorem 3.4. *Suppose $D = \{L_1, L_2, ..., L_r\}$ is a layer partitioning of a sample S. Assume this layer partitioning is equipped with a quality function $Q : D \to \mathbb{R}^+$ such that*

$$Q(L_i) \geq Q(L_j) \text{ as soon as } i \leq j. \tag{4}$$

Then the layer partitioning D and the quality Q induce a monotone combiner defined by its discriminant function

$$score(x) = Q(L_{l(x)}), \tag{5}$$

where the function $l : \mathbb{R}^k \to \{1, 2, ..., r\}$ *is defined by*

$$l(x) = \begin{cases} \min\{i : \exists y \in L_i,\ s.t.\ y \leq x\} & \text{if } \exists y \in S\ s.t.\ y \leq x, \\ r & \text{if } \nexists y \in S\ s.t.\ y \leq x. \end{cases} \tag{6}$$

Proof. Note that function $l(x)$ simply maps a point from the feature space to a layer with the best possible quality. We finish the proof observing that the prolongation $Q(L_{l(x)})$ of the quality function is non-decreasing with respect to x. □

Remark 3.5. In the rest of the paper we analyse monotone combiners in terms of layer partitioning. There is, however, an alternative approach. In order to describe a combiner we can use assignment matrices triggering the ranking of the sample points. In this case training a combiner is reduced to an assignment problem with an appropriate objective function, see [22]. We think, however, that our considerations are more perspicuous if formulated in terms of layers.

Remark 3.6. The choice of the prolongation provided by Theorem 3.1 is not unique. This issue was examined in detail in [23]. It has been shown that there is a minimum and a maximum prolongation, so that any prolongation between these two is also a valid one. Of course, the same applies also to Theorem 3.4.

Remark 3.7. If the range of values $[0, 1]$ is desired in a current application, then without violating their monotonicity we can scale the prolongations of the score and quality functions q from (2) and Q from (5) by means of tanh as follows:

$$score(x) = \tanh\left(q(s(x))\right)$$

in case of Theorem 3.1, and

$$score(x) = \tanh\left(Q(L_{l(x)})\right)$$

in case of Theorem 3.4.

Remark 3.8. The considerations of this section give us ideas about an opportunity to backtrack a combiner outcome according to introductory Section 1:

- *False positive*: a reason for a *positive* decision for the input point x can provided by (3) or (6); the point $\bar{t}_1$ on Figure 6 represents the reason why x has its score.
- *False negative*: *closest* candidates with higher scores can be searched for within the cone induced by the point under consideration; the points $\bar{t}_2$ and $\bar{t}_3$ on Figure 6 can be interpreted as the *closest* higher scored points with respect to x.

So far we have considered a monotone combiner and its properties, assuming that either a collection of point scores or a layers description is provided. However, the question arises as to how we can provide those scores or layers, and furthermore, what useful criteria they should fulfill.

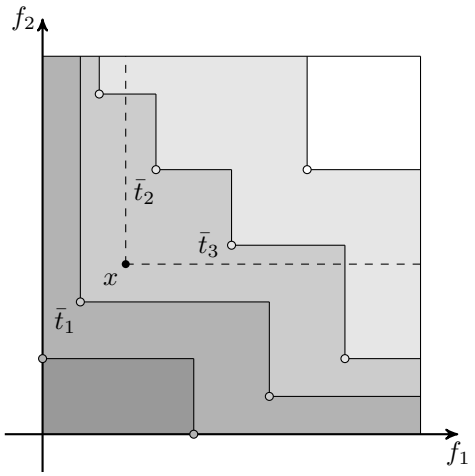


Figure 6: Backtracking an outcome for the input x: the point $\bar{t}_1$ represents the reason why x has its score; the points $\bar{t}_2$ and $\bar{t}_3$ are the *closest* representatives of the higher scored region.

Suppose S is a sample labelled in two classes. As before we call these classes *positives* and *negatives*. Let $P \subset S$ be a subsample with positively labelled and $N \subset S$ with negatively labelled individuals respectively. So we have $P \cup N = S$. The challenge is then to construct a monotone combiner based on this labelled sample. One possible approach can be based on a maximization of an AUC. In the next section we propose another approach based on a discrete trade-off analysis inspired by the Neyman-Pearson Lemma, see [9]. Later, in Corollary 6.5 of Section 6, we will show that both approaches indeed yield the same result.

4 Neyman-Pearson layers

Suppose likelihoods of *positives* and *negatives* are f_p and f_n respectively. In order to maximize the classifier strength the Neyman-Pearson Lemma suggests to prefer regions in the feature space with highest possible values of a ratio f_p/f_n. We refer to [24] for a comprehensive discussion.

Let us reformulate the Neyman-Pearson Lemma in terms of a constrained optimization problem. We consider

Optimization problem 4.1. Suppose $\alpha \in (0, 1)$. Find a region Ω in the feature space that solves

$$\underset{\Omega}{\text{maximize}} \quad \int_{\Omega} f_p(x)dx, \tag{7}$$

$$\text{subject to} \quad \int_{\Omega} f_n(x)dx \leq \alpha. \tag{8}$$

For the latter optimization problem we have

Lemma 4.2 (Neyman-Pearson)**.** *A region* Ω *defined by*

$$\Omega = \left\{ x : \frac{f_p(x)}{f_n(x)} \geq \eta \right\}$$

solves Optimization problem 4.1, where η *is the smallest possible value for which* (8) *still holds true.*

Let us interpret this lemma. If we are searching for a region inhabited with as many *positives* as possible, and are ready to accept only a certain amount of *negatives*, then Lemma 4.2 suggests to prefer subregions where the Neyman-Pearson ratio $f_p(x)/f_n(x)$ is as large as possible.

We are going to follow this suggestion in this section. Continuous versions of f_p and f_n are however both unknown and cannot be approximated in practice since the feature space is weakly inhabited in general, especially for larger dimension sizes k. On the other hand we can use a discrete analogue of the Neyman-Pearson ratio in order to construct a layer partitioning. We describe the construction in the form of Algorithm 1.

Algorithm 1: A naive algorithm driven by the Neyman-Pearson ratios

Input: Sample $P \cup N$, with P and N *positive* and *negative* counterparts.

Output: A layer partitioning $D = \{L_1, L_2, \ldots, L_r\}$.

Assign a still uncovered subsample $U := P \cup N$.
Assign a layer counter $r := 0$.
while $|U| > 0$ **do**
    Increase $r := r + 1$.
    Choose a subsample V of U, that maximizes the Neyman-Pearson quality among all monotone subsamples, i.e.

$$\text{find } V \subset U \text{ s.t. } \frac{|V \cap P|}{|V \cap N|} = \max_{\substack{W \subset U \\ W monotone}} \frac{|W \cap P|}{|W \cap N|}$$

    Assign V to be the next layer $L_r := V$.
    Update the uncovered subsample $U := U \setminus V$.

Remark 4.3. If the choice of the first layer L_1 is possible so, that $L_1 \cap N = \emptyset$, then we expect this layer to maximize the amount of positives $|L_1 \cap P|$.

The algorithm provides a layer partitioning $D = \{L_1, L_2, \ldots, L_r\}$ that we are going to study in detail. First we show that D induces a monotone combiner in the following

Theorem 4.4. *The layer partitioning $D = \{L_1, L_2, \ldots, L_r\}$ resulting from Algorithm 1 equipped with a quality function*

$$Q(L_i) = \frac{|L_i \cap P|}{|L_i \cap N|} \tag{9}$$

induces a monotone combiner.

Proof. The proof follows directly from the construction of the layer partitioning and Theorem 3.4. □

Algorithm 1 does not only define a monotone combiner, but the one possessing a useful optimality property similar to the Neyman-Pearson Lemma 4.2. To prove this, we will need a solid technical basis, which we will provide in the next four lemmas. We start with some very simple properties.

Lemma 4.5. *Let a, b, c, d be positive real numbers, then*

$$\text{if } \frac{a}{b} > \frac{c}{d}, \text{ then } \frac{a+c}{b+d} > \frac{c}{d}. \tag{10}$$

Let additionally $a \geq c$ and $b \geq d$, then

$$\text{if } \frac{a}{b} \geq \frac{c}{d}, \text{ then } \frac{a-c}{b-d} \geq \frac{a}{b}. \tag{11}$$

Assume also that positive real numbers a_i, b_i, $i = 1 \ldots n$ are provided, then

$$\text{if } \frac{a_i}{b_i} \geq \frac{c}{d} \ \forall i = 1 \ldots n, \text{ then } \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i} \geq \frac{c}{d}. \tag{12}$$

Proof. We prove the assertions of this lemma one by one.

$$\begin{aligned}
&\frac{a}{b} > \frac{c}{d} \Rightarrow ad > bc \Rightarrow ad + cd > bc + cd \\
&\qquad \Rightarrow (a+c)d > c(b+d) \Rightarrow \frac{a+c}{b+d} > \frac{c}{d} && \Rightarrow (10) \text{ holds;} \\
&\frac{a}{b} \geq \frac{c}{d} \Rightarrow ad \geq bc \Rightarrow -bc \geq -ad \Rightarrow ab - bc \geq ab - ad \\
&\qquad \Rightarrow (a-c)b \geq a(b-d) \Rightarrow \frac{a-c}{b-d} \geq \frac{a}{b} && \Rightarrow (11) \text{ holds;} \\
&\frac{a_i}{b_i} \geq \frac{c}{d} \Rightarrow a_i d \geq b_i c \Rightarrow \sum_{i=1}^n a_i d \geq \sum_{i=1}^n b_i c \\
&\qquad \Rightarrow d \sum_{i=1}^n a_i \geq c \sum_{i=1}^n b_i \Rightarrow \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i} \geq \frac{c}{d} && \Rightarrow (12) \text{ holds.}
\end{aligned}$$

□

Lemma 4.6. *Let A, B and C be arbitrary finite sets, then*

$$|A| = |A \setminus B| + |A \cap B|; \tag{13}$$
$$(A \setminus B) \cap (C \setminus B) = (A \setminus B) \cap C; \tag{14}$$
$$|A \cap C| = |(A \setminus B) \cap C| + |A \cap B \cap C|. \tag{15}$$

Proof. The identity (13) is evident. The identity (14) follows from the fact that it is redundant to subtract B from C while the result is then intersected with a set where instances from B are not present. We show (15) as follows

$$\begin{aligned} |A \cap C| &\overset{(13)}{=} |(A \cap C) \setminus B| + |A \cap B \cap C| \\ &= |(A \setminus B) \cap (C \setminus B)| + |A \cap B \cap C| \overset{(14)}{=} |(A \setminus B) \cap C| + |A \cap B \cap C| \end{aligned}$$

□

In order to continue we would need an additional definition.

Definition 4.7. Let A and B be subsamples (not necessarily monotone) of the sample $S = P \cup N$. We say that A dominates B and write $A \succeq B$ if

$$\frac{|A \cap P|}{|A \cap N|} \geq \frac{|B \cap P|}{|B \cap N|}.$$

In the notation of this definition Step 5 of Algorithm 1 creates a layer that dominates every other possible layer within the subsample of still available instances. We proceed with some basic properties of the defined relation $\succeq$.

Lemma 4.8. *Let A, B and C be subsamples (not necessarily monotone) of the sample $S = P \cup N$. The following property hold:*

$$\text{if } A \succeq B \text{ and } B \succeq C, \text{ then } A \succeq C \tag{16}$$

Assume also that a collection of pairwise disjoint subsamples A_i, $i = 1 \dots n$ of S is provided, then

$$\text{if } A_i \succeq B \ \forall\ i = 1 \dots n, \text{ then } A_1 \cup A_2 \cup \dots \cup A_n \succeq B. \tag{17}$$

Proof. The first property (16) follows directly from the Definition 4.7. We show (17) as follows.

$$\frac{|(A_1 \cup A_2 \cup \dots \cup A_n) \cap P|}{|(A_1 \cup A_2 \cup \dots \cup A_n) \cap N|} = \frac{|A_1 \cap P| + \dots + |A_n \cap P|}{|A_1 \cap N| + \dots + |A_n \cap N|} \overset{(12)}{\geq} \frac{|B \cap P|}{|B \cap N|}.$$

□

We now formulate some useful properties of layer partitioning created by the Algorithm 1.

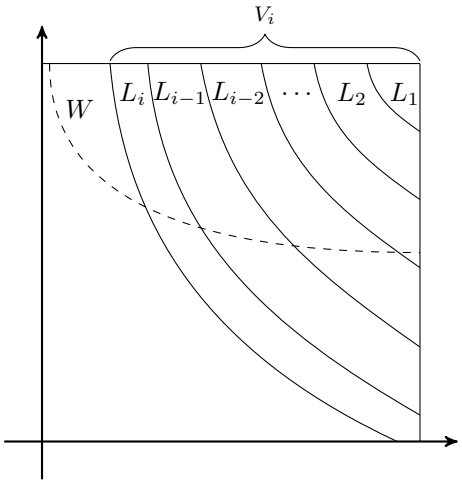


Figure 7: The layers $L_1, L_2, \ldots, L_i$ of the partitioning D and an alternative monotone subsample W.

Lemma 4.9. *Suppose $D = \{L_1, L_2, \ldots, L_r\}$ is a layer partitioning constructed by means of Algorithm 1 and W is an arbitrary monotone subsample of the sample $S = P \cup N$. Let $V_i = L_1 \cup L_2 \cup \ldots \cup L_i$, see Figure 7. The following properties hold:*

$$L_j \succeq L_{j+1}, \qquad \forall j \in [1, r-1]; \quad (18)$$

$$L_i \succeq L_i \cap W, \qquad \forall i \in [1, r]; \quad (19)$$

$$\text{if } L_i \setminus W \neq \emptyset, \text{ then } \quad L_i \setminus W \succeq L_i, \qquad \forall i \in [1, r]; \quad (20)$$

$$\text{if } W \setminus V_i \neq \emptyset, \text{ then } \quad L_i \succeq W \setminus V_i, \qquad \forall i \in [1, r]; \quad (21)$$

$$\text{if } L_j \setminus W \neq \emptyset \ \&\ W \setminus V_i \neq \emptyset, \text{ then } \quad L_j \setminus W \succeq W \setminus V_i, \qquad 1 \leq j \leq i \leq r; \quad (22)$$

$$\text{if } V_i \setminus W \neq \emptyset \ \&\ W \setminus V_i \neq \emptyset, \text{ then } \quad V_i \setminus W \succeq W \setminus V_i, \qquad \forall i \in [1, r]. \quad (23)$$

Proof. We prove (18) by contradiction. Assume (18) does not hold, i.e.

$$\frac{|L_j \cap P|}{|L_j \cap N|} < \frac{|L_{j+1} \cap P|}{|L_{j+1} \cap N|}. \quad (24)$$

Let us consider $\tilde{L} = L_j \cup L_{j+1}$, which is obviously a feasible layer in Algorithm 1 at iteration j, and thus provides an alternative to L_j. For $\tilde{L}$ we have

$$\frac{|\tilde{L} \cap P|}{|\tilde{L} \cap N|} = \frac{|(L_j \cup L_{j+1}) \cap P|}{|(L_j \cup L_{j+1}) \cap N|} = \frac{|L_j \cap P| + |L_{j+1} \cap P|}{|L_j \cap N| + |L_{j+1} \cap N|} \overset{(24),(10)}{>} \frac{|L_j \cap P|}{|L_j \cap N|},$$

which contradicts the choice of L_j in Algorithm 1. Thus our assumption (24) was wrong and (18) indeed holds. The next assertion (19) follows directly from

the choice of L_i in Algorithm 1. We prove (20) as follows

$$\frac{|(L_i \setminus W) \cap P|}{|(L_i \setminus W) \cap N|} \overset{(15)}{=} \frac{|L_i \cap P| - |L_i \cap W \cap P|}{|L_i \cap N| - |L_i \cap W \cap N|} \overset{(19),(11)}{\geq} \frac{|L_i \cap P|}{|L_i \cap N|}.$$

The proof of (21) is the same as that of (18), where $\tilde{L} = L_i \cup (W \setminus V_i)$ is chosen, and therefore omitted here. The assertion (22) follows directly from the previous results:

$$L_j \setminus W \overset{(20)}{\succeq} L_j \overset{(18)}{\succeq} L_i \overset{(21)}{\succeq} W \setminus V_i.$$

What remains is to show (23):

$$V_i \setminus W = (L_1 \cup L_2 \cup ... \cup L_i) \setminus W = (L_1 \setminus W) \cup (L_2 \setminus W) \cup ... \cup (L_i \setminus W) \overset{(22),(17)}{\succeq} W \setminus V_i.$$

□

We have made all the necessary preparations and are now ready to formulate a discrete optimization problem similar to the Optimization problem 4.1, for which we will prove a discrete analogue of the Neyman-Pearson Lemma 4.2.

Optimization problem 4.10. Suppose a sample $S = P \cup N$ is given and $0 \leq \alpha \leq |N|$. Find a monotone subsample $V \subset S$ that solves

$$\underset{\substack{V \subset S \\ V monotone}}{\text{maximize}} \quad |V \cap P|, \tag{25}$$

$$\text{subject to} \quad |V \cap N| \leq \alpha. \tag{26}$$

We continue with the main result of this section, which guarantees that for some discrete values of α, starting with zero up to $|N|$, Algorithm 1 provides us with an optimal solution for the latter optimization problem.

Lemma 4.11 (Discrete Neyman-Pearson lemma for monotone regions)**.** *Suppose $D = \{L_1, L_2, \ldots, L_r\}$ is a layer partitioning constructed by means of Algorithm 1. Then for each*

$$\alpha_i = |(L_1 \cup L_2 \cup ... \cup L_i) \cap N| \tag{27}$$

the solution of Optimization problem 4.10 reads

$$V_i = L_1 \cup L_2 \cup ... \cup L_i.$$

Proof. Suppose $W \subset S$ is an arbitrary monotone subsample, see Figure 7. We have to show that for any monotone $W \subset S$ satisfying

$$|W \cap N| \leq \alpha_i, \tag{28}$$

W satisfies $|W \cap P| \leq |V_i \cap P|$. This would mean that V_i indeed maximizes the objective function (25). For the proof we will require an additional property

$$|(W \setminus V_i) \cap N| \leq |(V_i \setminus W) \cap N|, \tag{29}$$

which we are going to show first; indeed:

$$\begin{aligned} |(W \setminus V_i) \cap N| &\overset{(15)}{=} |W \cap N| - |V_i \cap W \cap N| \overset{(28)}{\leq} \alpha_i - |V_i \cap W \cap N| \\ &\overset{(27)}{=} |V_i \cap N| - |V_i \cap W \cap N| \overset{(15)}{=} |(V_i \setminus W) \cap N|. \end{aligned}$$

We proceed by estimating $|W \cap P|$ from above as follows:

$$\begin{aligned} |W \cap P| &\overset{(15)}{=} |(W \setminus V_i) \cap P| + |V_i \cap W \cap N| \\ &= \frac{|(W \setminus V_i) \cap P|}{|(W \setminus V_i) \cap N|} |(W \setminus V_i) \cap N| + |V_i \cap W \cap N| \\ &\overset{(23)}{\leq} \frac{|(V_i \setminus W) \cap P|}{|(V_i \setminus W) \cap N|} |(W \setminus V_i) \cap N| + |V_i \cap W \cap N| \\ &\overset{(29)}{\leq} \frac{|(V_i \setminus W) \cap P|}{|(V_i \setminus W) \cap N|} |(V_i \setminus W) \cap N| + |V_i \cap W \cap N| \\ &= |(V_i \setminus W) \cap P| + |V_i \cap W \cap N| \\ &\overset{(15)}{=} |V_i \cap P|. \end{aligned}$$

What remains is to check the special cases, which are not covered by (23):

- $W \setminus V_i = \emptyset$, which leads obviously to $|W \cap P| \leq |V_i \cap P|$;
- $V_i \setminus W = \emptyset \Rightarrow V_i \subset W \overset{(27),(28)}{\Longrightarrow} W \setminus V_i \subset P$, which contradicts the choice of L_i and thus cannot happen.

□

Lemma 4.11 is of great importance for us. It ensures the optimality of Algorithm 1 for $\alpha \in [0, |N|]$ discretized with (27). For some $\alpha \in [0, |N|]$ lying strictly between sequential α_i and α_{i+1} an example can be provided, for which an optimal subsample $V \subset S$ satisfying (25) does not lie between V_i and V_{i+1}. Lemma 4.11 guarantees, however, that the objective function of this solution $|V \cap P|$ goes to $|V_i \cap P|$ as α increases to α_{i+1}.

Up to now we have constructed a monotone combiner by means of the Neyman-Pearson quality measure. However, in order to utilize this combiner we have to address two following issues first.

- Evidently the choice of the next layer may be non-unique at some steps. Therefore is the layer partitioning unique in any sense?
- Directly applying Algorithm 1 would yield an exponential behaviour in the sample size. However, can we provide a polynomial algorithm to construct this layer partitioning in practice?

Fortunately, we are able to resolve both issues in the way shown in the following sections.

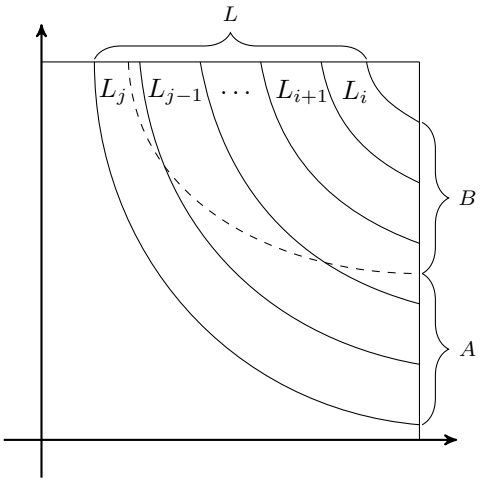


Figure 8: The layers $L_i, L_{i+1}, \ldots, L_j$ of the partitioning D merged into one layer L. $\{A, B\}$ is an alternative partitioning of L.

5 Uniqueness

We start with a definition. As before, suppose a sample $S = P \cup N$ is formed by two subsamples, where P represents the *positives* and N represents the *negatives*.

Definition 5.1 (Neyman-Pearson layer partitioning)**.**
A partitioning $D = \{L_1, L_2, ..., L_r\}$ of a sample $S = P \cup N$ is called a Neyman-Pearson partitioning if the Neyman-Pearson ratio of each subsequent layer does not exceed the one from the previous layer, i.e.

$$\forall i \in [2, r] \qquad \frac{|L_{i-1} \cap P|}{|L_{i-1} \cap N|} \geq \frac{|L_i \cap P|}{|L_i \cap N|}. \tag{30}$$

We call this partitioning the *finest* Neyman-Pearson partitioning if each layer L_i does not itself allow its own Neyman-Pearson partitioning involving a strict inequality sign $>$ in (30). In this case we say that the layer L_i is not decomposable in the Neyman-Pearson sense.

Note that in Algorithm 1 we come up with one of the possible *finest* Neyman-Pearson partitionings. We are going to show that two different finest Neyman-Pearson partitionings differ only in layers of same quality. If they are merged a unique final Neyman-Pearson partitioning is reached. To this end we need the following

Lemma 5.2. *Assume $D = \{L_1, L_2, ..., L_r\}$ is a finest Neyman-Pearson partitioning. If we merge layers of the same quality within the partitioning then the result is also a finest Neyman-Pearson partitioning.*

Proof. The resulting partitioning is evidently Neyman-Pearson. What remains is to show that it is *finest*. Consider the layers $L_i, L_{i+1}, \ldots, L_j$ of the same quality, that merge into one layer $L = L_i \cup L_{i+1} \cup ... \cup L_j$, see Figure 8. Assume that the resulting layer is decomposable into at least two Neyman-Pearson layers A and B of different qualities

$$\frac{|A \cap P|}{|A \cap N|} < \frac{|B \cap P|}{|B \cap N|},$$

where B is a monotone subsample of L. Among others this means

$$\frac{|A \cap P|}{|A \cap N|} < \frac{|L \cap P|}{|L \cap N|} < \frac{|B \cap P|}{|B \cap N|}. \tag{31}$$

Note that it is enough to consider a two layer case since any multiple layer case can be reduced to two layers by subsequently merging adjacent layers.

Inequality (31) implies that one of the layers $L_i, L_{i+1}, \ldots, L_j$ has a monotone subsample of quality better than $|L \cap P|/|L \cap N|$; this layer can thus be decomposed into two Neyman-Pearson layers of different quality. The last argument contradicts the assumption of the *finest* Neyman-Pearson partitioning, which completes the proof. □

Now we are ready to show the main result of this section.

Theorem 5.3 (Uniqueness)**.** *Suppose two finest Neyman-Pearson partitionings are given. If we merge layers of the same quality, the resulting partitionings will coincide.*

Proof. Assume after merging layers of the same quality the partitionings $D_1 = \{L_1, L_2, ..., L_{r_1}\}$ and $D_2 = \{M_1, M_2, ..., M_{r_2}\}$ do not coincide. Choose the first non-coinciding layers of the partitionings $L_j \in D_1$ and $M_j \in D_2$, i.e.

$$L_j \neq M_j \text{ and } L_i = M_i \quad \forall i < j. \tag{32}$$

Note that L_j and M_j are of the same quality, otherwise their "up right" parts before merging would be of different quality.

Without loss of generality we may assume $\Omega = L_j \setminus M_j \neq \emptyset$. There are three possibilities:

1. Ω is of better quality than L_j and M_j. Then M_j can be made better by Ω, which contradicts the *finest* condition of D_2 guaranteed by Lemma 5.2.

2. Ω is of the same quality as L_j and M_j. This is also not possible due to arguments similar to bullet 1.

3. Ω is of lower quality than L_j and M_j. Then L_j can be made better by removing Ω, which contradicts the *finest* condition of D_1 guaranteed by Lemma 5.2.

All three possibilities cannot hold. Thus our assumption is not true, which finishes the proof. □

6 Polynomial time construction algorithm Part I

Up to this point we have shown that the *finest* Neyman-Pearson partitioning is unique as soon as we merge parts of the same quality. It means that Algorithm 1 also results in this unique partitioning. Moreover, any procedure producing the *finest* Neyman-Pearson partitioning would yield the same result. In this and the next sections we provide an approach to construct the *finest* Neyman-Pearson partitioning in a polynomial time of the sample size $|P \cup N|$. Our approach is based on sequentially splitting layers that are decomposable in the Neyman-Pearson sense and merging adjacent layers in case of Neyman-Pearson inconsistencies.

In order to make the presentation simpler we divide the algorithm into two logically independent parts:

- the first part is Algorithm 2 below. It iteratively constructs a layer partitioning of the sample $P \cup N$. Later we will show that the iterations converge to the *finest* Neyman-Pearson partitioning in a finite number of steps.
- the second part is Algorithm 3 in the next section. This algorithm is used at a particular step within Algorithm 2 in order to split a layer into two layers in case it is decomposable in the Neyman-Pearson sense.

Consider Algorithm 2. We focus on this algorithm for the rest of the section.

Remark 6.1. Mention that the algorithm does not prescribe which of conditions 3 or 5 should be checked first. The choice is left to the user.

Remark 6.2. Mention that if the algorithm converges, then it converges against the *finest* Neyman-Pearson partitioning due to its definition.

In order to prove the convergence of Algorithm 2 and show its convergence rate, it is convenient to recall ROC analysis. For the extensive introduction into ROC curves and an area under the curve concept we refer to [25]. Here we are going to use the fact first described in [10], namely

Lemma 6.3. *The area under the curve (AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.*

Now we are ready to prove the convergence result.

Theorem 6.4. *For Algorithm 2 we have*

1. *the AUC corresponding to the classifier induced by the current layer partitioning D increases at each successful iteration of Loop 1-6 of Algorithm 2;*

Algorithm 2: *Finest* Neyman-Pearson partitioning of a sample

Input: Sample $P \cup N$, with P and N *positive* and *negative* counterparts.

Output: *Finest* Neyman-Pearson partitioning $D = \{L_1, L_2, \ldots, L_r\}$.

1 Assign a current layer partitioning D to contain the only layer $P \cup N$, i.e.

$$D := \{P \cup N\}$$

while *D changes* **do**

2 Try one of the if-blocks of your choice below. If its condition fails, try another block.

3 **if** *there is a layer L_j which can be split into two layers L^+, L^- in the Neyman-Pearson sense, i.e. for a proper monotone subsample $L^+ \subset L_j$ and $L^- = L_j \setminus L^+$ holds*

$$\frac{|L^+ \cap P|}{|L^+ \cap N|} > \frac{|L^- \cap P|}{|L^- \cap N|},$$

then

4 replace L_j via two subsequent layers L^+, L^- within the current partitioning D.

5 **if** *in the current partitioning D there are two adjacent layers L_j and L_{j+1} that are inconsistent in the Neyman-Person sense, i.e.*

$$\frac{|L_j \cap P|}{|L_j \cap N|} < \frac{|L_{j+1} \cap P|}{|L_{j+1} \cap N|},$$

then

6 merge them into one layer $\widetilde{L} = L_j \cup L_{j+1}$ and replace these two layers L_j, L_{j+1} via $\widetilde{L}$ within the current partitioning D.

2. *Algorithm 2 converges in a finite number of steps. Moreover,*

3. *if Block 3-4 can be done in $s(q)$ operations, then the whole Algorithm 2 converges in no more than $s(q)O(q^3)$ operations, where $q = |P \cup N|$ is the sample size.*

Proof. In our case the classifier induced by $D = \{L_1, L_2, \ldots, L_r\}$ is applied to the sample $P \cup N$. The corresponding AUC can be computed using Lemma 6.3 via

$$AUC = 1 - \frac{|\textit{ranking errors}|}{|P||N|}.$$

Due to this observation, bullet 1 of the theorem is equivalent to a decrease of the *ranking errors* at each successful iteration of Loop 1-6 of Algorithm 2. Let us consider two cases: 3-4 is successful and 5-6 is successful. Notice that in both cases it is enough to consider the change of the *ranking errors* only within the layers being modified, since other *ranking errors* will remain constant as we apply one iteration of Loop 1-6.

3-4. In this case there is a layer L_j in the current partitioning D which is decomposable in the Neyman-Pearson sense $L_j = L^+ \cup L^-$ with $L^+ \cap L^- = \emptyset$ and

$$\frac{|L^+ \cap P|}{|L^+ \cap N|} > \frac{|L^- \cap P|}{|L^- \cap N|}. \tag{33}$$

We use the following notation:

$$\begin{aligned} a &= |L^+ \cap P|, \\ b &= |L^+ \cap N|, \\ c &= |L^- \cap P|, \\ d &= |L^- \cap N| \end{aligned}$$

Mention that with this notation $|L_j \cap P| = a + c$, $|L_j \cap N| = b + d$. On the other hand we can rewrite (33) as

$$\frac{c}{d} < \frac{a}{b}, \text{ or } cb < ad. \tag{34}$$

The amount of the *ranking errors* within L_j before splitting is

$$errors(L_j) = \frac{(a+c)(b+d)}{2} = \frac{ab}{2} + \frac{cd}{2} + \frac{cb}{2} + \frac{ad}{2}.$$

Here and later we are operating with an assumption that points from the sample are randomly ranked as long as they are sharing one layer. After the splitting of L_j into $\{L^+, L^-\}$, the *ranking errors* within L_j become

$$errors(L^+, L^-) = \frac{ab}{2} + \frac{cd}{2} + cb.$$

The change of the *ranking errors* is then

$$errors(L^+, L^-) - errors(L_j) = \frac{cb - ad}{2} < 0, \tag{35}$$

due to (34). Or in other words the *ranking errors* decrease, thus the AUC increases.

5-6. Let two adjacent layers L_j and L_{j+1} in the current partitioning D be inconsistent in the Neyman-Person sense. We follow the same lines as in case 3-4. Use the notation:

$$\begin{aligned} a &= |L_j \cap P|, \\ b &= |L_j \cap N|, \\ c &= |L_{j+1} \cap P|, \\ d &= |L_{j+1} \cap N|. \end{aligned}$$

Then the inconsistency in the Neyman-Pearson sense means

$$\frac{c}{d} > \frac{a}{b}, \text{ or } cb > ad. \tag{36}$$

The *ranking errors* within the merged layer are:

$$errors(L_j \cup L_{j+1}) = \frac{(a+c)(b+d)}{2} = \frac{ab}{2} + \frac{cd}{2} + \frac{cb}{2} + \frac{ad}{2}$$

The *ranking errors* within two layers before merging were:

$$errors(L_j, L_{j+1}) = \frac{ab}{2} + \frac{cd}{2} + cb.$$

And thus

$$errors(L_j \cup L_{j+1}) - errors(L_j, L_{j+1}) = \frac{ad - cb}{2} < 0, \tag{37}$$

due to (36). Therefore the proof of bullet 1 is complete.

In order to show bullet 2 we first notice that in both Blocks 3-4 and 5-6 of Algorithm 2 the amount of *ranking errors* reduces by at least $\frac{1}{2}$ due to (35) and (37). On the other hand the amount of the *ranking errors* is obviously bounded from below by zero. Thus the number of successful execution of Loop 1-6 is finite, which implies bullet 2. Moreover, this number is at most $O(q^2)$, because the amount of the *ranking errors* is bounded from above by $|P| \times |N|$. Furthermore, we can estimate from above the search of layers satisfying conditions of 3 or 5 by $O(q)$. Using the cost estimation $s(q)$ for Block 3-4 of the algorithm we finish the proof. □

Corollary 6.5. *The AUC of the induced combiner is maximized via the finest Neyman-Pearson layer partitioning.*

Proof. Assume the maximization of AUC does not yield the *finest* Neyman-Pearson layer partitioning. Then there are either two layers conflicting in the Neyman-Person sense or at least one decomposable layer. As we know from the previous theorem joining or splitting the layers respectively would lead to a larger AUC, which contradicts the assumption. □

Besides the fact that we have shown the convergence of Algorithm 2 we have also built a bridge between the Neyman-Pearson and the ROC curve quality measures for the monotone combiners. We have shown, namely, that the algorithm for the *finest* Neyman-Pearson partitioning also maximizes the AUC of the induced combiner.

7 Polynomial time construction algorithm Part II

In this section we address Block 3-4 of Algorithm 2 of the previous section. Given a layer we show how we can check whether this layer is decomposable in the Neyman-Pearson sense and if applicable provide the corresponding partitioning. In addition we show that our algorithm is of complexity at most $O(|L|^3)$, where $|L|$ is the layer size.

We proceed as follows. First we reduce the problem to a linear program. This linear program however appears to be essentially degenerating and we are unable to estimate its complexity in a general case; although one step of the simplex method would be enough for our purpose if the problem was non-degenerating. In order to overcome this difficulty, we take a dual to the latter linear program. After some additional manipulation the dual problem fortunately appears to be equivalent to a certain maximum flow problem. There are some very efficient algorithms [26, 27, 28, 29] that cope with this problem. After solving the dual problem we use complementary slackness conditions in order to calculate the solution of the original linear program. In turn this solution provides us with the answer as to whether the layer is decomposable in the Neyman-Pearson sense and if so, then explicitly what the partitioning is.

Without loss of generality we assume that the layer under consideration contains all samples from the feature space $L = P \cup N$, where P represents *positives* and N represents *negatives*. We can formalize the decomposability condition of a layer with the help of the following discrete optimization problem. Namely, a layer L is decomposable if and only if the following problem yields a solution.

Optimization problem 7.1. Find a proper monotone subset $L^+ \subset L$ with a better Neyman-Pearson quality than L itself, i.e.

$$\frac{|L^+ \cap P|}{|L^+ \cap N|} > \frac{|L \cap P|}{|L \cap N|}.$$

In this case the quality of L^+ would obviously also be greater than the quality of the rest $L^- = L \setminus L^+$. The desired Neyman-Pearson partitioning would then be $\{L^+, L^-\}$.

In order to avoid redundant variables and conditions, we consolidate identical points within our sample $L = P \cup N$. Namely, we consider a set of points

$V = \{v_i\} \subset \mathbb{R}^k$ equipped with two multiplicity functions

$$\begin{aligned} p(v_i) &= \text{"Multiplicity of } v_i \text{ in } P\text{"} = |\{v_i\} \cap P|, \\ n(v_i) &= \text{"Multiplicity of } v_i \text{ in } N\text{"} = |\{v_i\} \cap N|, \end{aligned}$$

satisfying

$$v_i \in P \cup N \text{ implies } v_i \in V \text{ and } \forall v_i, v_j \in V \quad v_i = v_j \text{ implies } i = j.$$

Notice that the following relations hold:

$$\sum_i p(v_i) = |P| \text{ and } \sum_i n(v_i) = |N|.$$

In order to handle the monotone property of the desired subsample L^+ we will use the following auxiliary graph.

Definition 7.2 (Graph induced by $\leq$)**.** We define graph $G = (V, E)$ having a set of nodes $V = \{v_i\}$ and a set of edges induced by the relation $\leq$, i.e. there is an edge $(i, j) \in E$ if and only if $v_i \leq v_j$.

Remark 7.3. It is easy to see that the construction of this graph takes $O(|V|^2)$ processing time. We could also do a transitive reduction of this graph to remove unnecessary edges, which in practice would mean fewer edges $|E|$ for the subsequent computations. The transitive reduction would however also mean an additional processing time of order $O(|V|^3)$.

With the help of the auxiliary graph G we reformulate Optimization problem 7.1 in the following form.

Optimization problem 7.4. Find $x_i \in \{0, 1\}$ such that

$$\begin{aligned} \frac{\sum_i x_i p(v_i)}{\sum_i x_i n(v_i)} &> \frac{|P|}{|N|}, \text{ subject to} \\ x_i &\leq x_j \quad \forall (i, j) \in E. \end{aligned} \tag{38}$$

Notice, that

- if all the variables $x_i = 1$ then the inequality (38) turns into equality,
- if Optimization problem 7.4 has a solution then the desired L^+ consists of points corresponding to the variables $x_i = 1$,
- if we searched for a maximum of the left hand side of (38), we would come up with the best possible layer L^+.

We transform Optimization problem 7.4 into 7.5 below and then show that they are equivalent.

Optimization problem 7.5. Find $x_i \in \{0,1\}$, such that

$$|N| \sum_i x_i p(v_i) - |P| \sum_i x_i n(v_i) \quad > \quad 0, \text{ subject to} \tag{39}$$

$$x_i \quad \leq \quad x_j \quad \forall (i,j) \in E. \tag{40}$$

Lemma 7.6. *A collection of x_i solves Optimization problem 7.4 if and only if it solves Optimization problem 7.5.*

Proof. The proof is straight forward. □

Furthermore, what we will actually solve is a linear program which is a stronger form of Optimization problem 7.5:

Linear program 7.7. Find $x_i \in [0,1]$ that solves

$$\underset{x}{\text{maximize}} \quad Z(x) = |N| \sum_i x_i p(v_i) - |P| \sum_i x_i n(v_i), \tag{41}$$

$$\text{subject to} \quad 0 \leq x_i \leq 1, \tag{42}$$

$$x_i \leq x_j \quad \forall (i,j) \in E. \tag{43}$$

Lemma 7.8. *Optimization problem 7.5 has a solution if and only if Linear program 7.7 admits a solution $x_{\max}$ with an objective function value $Z(x_{\max}) > 0$.*

Proof. Proof is obvious except for the fact that each x_i is now a continuous variable in $[0,1]$. This is fortunately not an issue since a general linear program is known to yield a solution on one of the polytope vertices defined by constraints (42) and (43), see [30, 31]. Every such vertex fulfills a linear system of equations with an invertible matrix formed by the rows of the matrix corresponding to constraints (42) and (43). The equations in this linear system can be of three following types: either $x_i = 0$, $x_i = 1$ or $x_i = x_j$. Since the matrix of this system is invertible, the solution is unique and consists only of those x_i, that satisfy either $x_i = 0$ or $x_i = 1$. □

To illustrate the consideration so far and for the ease of further arguments we proceed with an example.

7.1 Example

Consider a layer $L = P \cup N$ with

$$P = \left\{ \left(\frac{1}{4}, \frac{1}{4}\right), \left(\frac{1}{4}, \frac{1}{4}\right), \left(\frac{1}{3}, \frac{2}{3}\right), \left(\frac{1}{3}, \frac{2}{3}\right) \right\} \text{ and}$$

$$N = \left\{ \left(\frac{1}{3}, \frac{2}{3}\right), \left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{2}{3}, \frac{1}{3}\right), \left(\frac{3}{4}, \frac{3}{4}\right) \right\}.$$

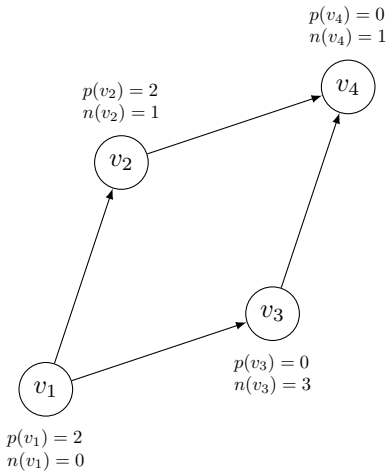


Figure 9: Sample in the feature space represented as a graph G induced by the relation $\leq$. The multiplicities of the classes are depicted at each node v_i.

For this layer we have

$$v_1 = \left(\frac{1}{4}, \frac{1}{4}\right), v_2 = \left(\frac{1}{3}, \frac{2}{3}\right), v_3 = \left(\frac{2}{3}, \frac{1}{3}\right), v_4 = \left(\frac{3}{4}, \frac{3}{4}\right)$$

with the corresponding multiplicities, see Figure 9. Note that the edge $(1, 4)$ is omitted in order to avoid an unnecessary constraint in the example. In other words we have performed the transitive reduction mentioned above.

For this example Linear program 7.7 reads:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & Z(x) = 5(2x_1 + 2x_2) - 4(x_2 + 3x_3 + x_4) \\ & = 10x_1 + 6x_2 - 12x_3 - 4x_4, \\ \text{subject to} \quad & x_1 \leq x_2, \\ & x_1 \leq x_3, \\ & x_2 \leq x_4, \\ & x_3 \leq x_4, \\ & 0 \leq x_1 \leq 1, \\ & 0 \leq x_2 \leq 1, \\ & 0 \leq x_3 \leq 1, \\ & 0 \leq x_4 \leq 1. \end{aligned} \tag{44}$$

Note that $Z((1, 1, 1, 1)^\top) = 0$, i.e. the sum of the coefficients of the objective function is 0. It will always be the case in a general situation since the inequality (38) turns into equality as soon as all $x_i = 0$ as already mentioned. The point $x = (1, 1, 1, 1)^\top$ corresponds to the layer $L^+ = L$ and is thus not of interest

to us. What is worth checking is whether there exists a point $x \neq (1,1,1,1)^\top$ with $Z(x) > 0$. To this end we could start with a feasible point $x = (1,1,1,1)^\top$ and perform a step of the simplex method. As can be seen, the problem (44) is degenerate. Thus a single step of the simplex method will not guarantee a better value of the objective function. There are strategies dealing with degenerate linear programs, see e.g. [32]. If we try to solve this problem directly, we do not however see an opportunity to avoid worst case exponential time behaviour in achieving a solution with $Z(x) > 0$.

Fortunately we are able to solve its dual in polynomial time. For the dual formulation we use a standard technique. To this end we introduce two collections of dual variables: β_i are the node associated and γ_{ij} are the edge associated variables. Herewith we get the following dual formulation:

$$\begin{aligned} \underset{\beta,\gamma}{\text{minimize}} \quad & Z^*(\beta,\gamma) = \beta_1 + \beta_2 + \beta_3 + \beta_4, \\ \text{subject to} \quad & \gamma_{12} + \gamma_{13} + \beta_1 \geq 10, \\ & -\gamma_{12} + \gamma_{24} + \beta_2 \geq 6, \\ & -\gamma_{13} + \gamma_{34} + \beta_3 \geq -12, \\ & -\gamma_{24} - \gamma_{34} + \beta_4 \geq -4, \\ & \beta_i \geq 0, \\ & \gamma_j \geq 0. \end{aligned} \tag{45}$$

The latter is nothing but a special flow problem, which we would interpret as follows. Each inequality constraint corresponding to a certain node in the graph tells us that an inflow to this node may not be larger than a potential outflow of this node. We have two inflow types:

- constant inflows: 10 for v_1, 6 for v_3, and
- variable inflows coinciding with outflows from other nodes, e.g. γ_{34} is an inflow of the node v_4 and an outflow of v_3.

At the same time we have three outflow types:

- constant potential outflows: 12 for v_3, 4 for v_4,
- variable outflows γ_{ij} coinciding with inflows for other nodes,
- additional possible outflows β_i which we would interpret as *helping* outflows.

With this semantic the linear program (45) can be reformulated as:
Find a combination of helping outflows β_i with a smallest possible sum for which there exist γ_{ij}, such that outflows at each node suffice its inflows.

In order to handle the constant potential inflows and outflows we mentioned above we enrich the graph G in the following way. We add two artificial nodes: a source s and a sink t, and artificial edges representing the constant inflows and outflows, see Figure 10. We denote the enriched graph by $\tilde{G}$. It is easy

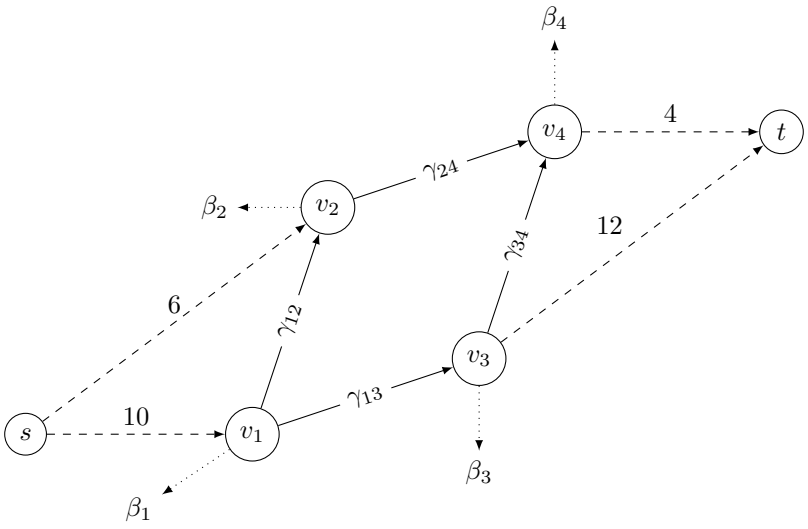


Figure 10: Enriched graph $\tilde{G}$; source s, sink t and dashed edges are added. Artificial edges are dashed and *helping* flows are dotted.

to see that it does not matter where to apply *helping* outflows β_i, upstream or downstream. We assume they are applied upstream. Thereby we can reinterpret the role of *helping* flows β_i as reducing the constant inflows in such a way that the remaining flow can be transported until the sink. Therefore, minimizing the *helping* flow means maximizing the flow in $\tilde{G}$ with

- maximum constraints on the artificial edges and
- no constraints on the original edges of the graph G.

This interpretation is the classical maximum flow problem.

Remark 7.9. We consider a general case later, where we provide a rigorous analysis of the relationship between our problem (45) and the corresponding maximum flow problem.

For our current example the maximum flow problem yields the solution depicted in Figure 11. Based on this solution we have:

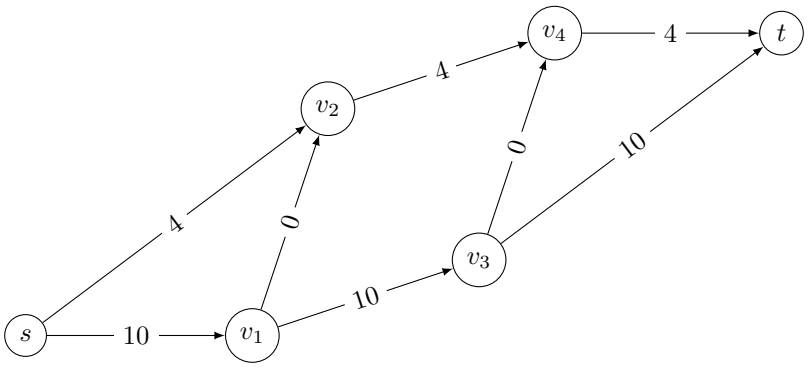


Figure 11: Maximal flow through $\tilde{G}$.

$$\begin{aligned}
\beta_1 &= 0, \\
\beta_2 &= 6 - 4 = 2, \\
\beta_3 &= 0, \\
\beta_4 &= 0, \\
\gamma_{12} &= 0, \\
\gamma_{13} &= 10, \\
\gamma_{24} &= 4, \\
\gamma_{34} &= 0.
\end{aligned}$$

Using the complementary slackness conditions we find the solution of the original problem:

$$\begin{aligned}
\beta_2 = 2 \neq 0 &\quad \rightarrow \quad x_2 = 1, \\
\gamma_{13} = 10 \neq 0 &\quad \rightarrow \quad x_1 = x_3, \\
\gamma_{24} = 4 \neq 0 &\quad \rightarrow \quad x_2 = x_4, \\
-\gamma_{13} + \gamma_{34} + \beta_3 = -10 > -12 &\quad \rightarrow \quad x_3 = 0.
\end{aligned}$$

So, the solution of (44) reads

$$\begin{aligned}
x_1 = x_3 = 0, \\
x_2 = x_4 = 1,
\end{aligned}$$

for which the corresponding layer partitioning has to be the Neyman-Pearson one. Let us check that the Neyman-Pearson ratio has increased from $L = \{v_1, v_2, v_3, v_4\}$ to $L^+ = \{v_2, v_4\}$. Indeed,

$$\frac{|L \cap P|}{|L \cap N|} = \frac{4}{5} < \frac{2}{2} = \frac{|L^+ \cap P|}{|L^+ \cap N|}.$$

Note also, that the Neyman-Pearson ratio decreases for the remaining part

$L^- = L \setminus L^+$ to

$$\frac{|L^- \cap P|}{|L^- \cap N|} = \frac{2}{3} < \frac{4}{5}.$$

7.2 General case

Consider now a general case. We rewrite Linear program 7.7 as

Linear program 7.10. Find $x_i \in [0,1]$, such that

$$\begin{aligned} \underset{x}{\text{maximize}} \quad Z(x) &= \sum_i c_i x_i, \\ \text{subject to} \quad 0 &\leq x_i \leq 1, \\ x_i &\leq x_j \quad \forall (i,j) \in E, \text{ with} \\ Z((1,1,1,...,1)^\top) &= 0, \text{ i.e. } \sum_i c_i = 0. \end{aligned}$$

Let us also consider a dual formulation of this problem. As in the example above we have two collections of dual variables: β_i are the vertex associated variables and γ_{ij} are the edge associated variables. The dual problem reads

Linear program 7.11.

$$\underset{\beta,\gamma}{\text{minimize}} \quad Z^*(\beta,\gamma) = \sum_i \beta_i, \quad \text{subject to}$$

$$-\sum_{j:(j,i)\in E} \gamma_{ji} + \sum_{j:(i,j)\in E} \gamma_{ij} + \beta_i \geq c_i, \tag{46}$$

$$\beta_i \geq 0, \tag{47}$$

$$\gamma_{ij} \geq 0. \tag{48}$$

The latter looks very much like a flow problem. We indeed formulate a maximum flow problem of a special kind and show that as soon as we know the solution of this maximum flow problem we can reconstruct one of the possible solutions of Linear program 7.11. We enrich the graph G with:

- two additional vertices s and t representing a source and a sink of the future maximum flow problem,
- $\forall i : c_i > 0$ we add an edge (s,i),
- $\forall i : c_i < 0$ we add an edge (i,t).

We denote the resulting set of vertices via $\tilde{V}$, the resulting set of edges via $\tilde{E}$ and the enriched graph via $\tilde{G} = (\tilde{V}, \tilde{E})$.

Now we are at the position to formulate the maximum flow problem. Let $f : \tilde{E} \to \mathbb{R}^+$ be a flow in the graph $\tilde{G}$. For the sake of better clarity we denote $f_{ij} = f(i,j)$. The desired maximum flow problem reads as follows.

Maximum flow problem 7.12. Find $f : \tilde{E} \to \mathbb{R}^+$ that solves

$$\begin{aligned}
\underset{f}{\text{maximize}} \quad F(f) &= \sum_{i:(s,i)\in\tilde{E}} f_{si}, \quad \text{subject to} \\
\text{conservation conditions:} \quad \sum_{j:(j,i)\in\tilde{E}} f_{ji} &= \sum_{j:(i,j)\in\tilde{E}} f_{ij} \quad \forall i \neq s,t, \qquad (49) \\
\text{capacity conditions:} \quad f_{si} &\leq c_i \quad \forall i : c_i > 0, \qquad (50) \\
\text{capacity conditions:} \quad f_{it} &\leq -c_i \quad \forall i : c_i < 0. \qquad (51)
\end{aligned}$$

In order to be able to prove the theorem showing the relationship between Maximum flow problem 7.12 and Linear program 7.11 we need the following

Lemma 7.13. *Let (β, γ) satisfy the the constraints (46, 47, 48). Then there exists a flow $f : \tilde{E} \to \mathbb{R}^+$ satisfying (49, 50, 51) with the flow value*

$$F(f) \geq \sum_{i:c_i>0} c_i - \sum_i \beta_i.$$

Proof. We give a constructive proof. We are going to manipulate β_i, γ_{ij} and delete some edges of G without disturbing constraints (46, 47, 48) and without increasing the objective function Z^*.

Step 1. Delete all the edges $(i, j) \in E$ with $\gamma_{ij} = 0$, or strictly speaking proceed as if those edges were not present.

Step 2. In this step we address those $\beta_i > 0$ that can be transported upstream. To this end check if there exists a path p in G, such that

$$\begin{aligned}
p \text{ starts at } m: &\quad c_m > \beta_m \geq 0, \\
p \text{ ends at } i: &\quad c_i \leq 0 \text{ and } \beta_i > 0.
\end{aligned} \qquad (52)$$

If no, we proceed with the next step. Otherwise consider a minimal flow on this path with a flow value $\gamma_{min(p)} = \min_{(i,j)\in p} \gamma_{ij}$. Note that $\gamma_{min(p)} > 0$ since we have eliminated all the edges with zero flow in Step 1. Let $w = min(c_m - \beta_m, \beta_i, \gamma_{min(p)})$. Modify:

$$\begin{aligned}
\beta_m &:= \beta_m + w, \\
\gamma_{ij} &:= \gamma_{ij} - w, \quad \forall (i,j) \in p, \\
\beta_i &:= \beta_i - w.
\end{aligned} \qquad (53)$$

As in Step 1 delete all the edges $(i, j) \in E$ with $\gamma_{ij} = 0$. Note that constraints (46, 47, 48) remain fulfilled and the value of the objective function Z^* does not change. Finally, repeat modification (53) as long as there exists a path p satisfying (52). After a finite number of iterations we will finish this step since the number of paths satisfying conditions (52) decreases every time we apply modification (53).

Step 3. We address eliminating $\beta_i > 0$ with $c_i \leq 0$. Assume that there is no path p that fulfills (52). In other words there is no unsatisfied node on the way to i upstream. Modify:

$$\begin{aligned} \beta_i &:= 0, \\ \beta_m &:= \max(0, c_m), \qquad \forall m \text{ lying on a path leading to } i, \\ \gamma_{mj} &:= 0, \qquad \forall (m,j) \text{ lying on a path leading to } i. \end{aligned} \tag{54}$$

Perform this modification for all $\beta_i > 0$ with $c_i \leq 0$. Note that constraints (46, 47, 48) remain fulfilled and the value of the objective function Z^* decreases.

Step 4. We assume that there is no i with $\beta_i > 0$ and $c_i \leq 0$. We start with source nodes and go downstream by successively reducing outflows at each node i in the following manner:

- for i: $c_i > 0$ we reach the equal sign in inequality constraint (46) by applying successfully

$$\begin{aligned} \sum_{j:(i,j)\in E} \gamma_{ij} &:= \max(0, c_i - \beta_i + \sum_{j:(j,i)\in E} \gamma_{ji}) \text{ and then} \\ \beta_i &:= \min(\beta_i, c_i + \sum_{j:(j,i)\in E} \gamma_{ji}); \end{aligned}$$

  note that the second assignment applies only in the case of a zero right hand side of the first assignment; inequality (46) indeed turns into equality;
- for i: $c_i \leq 0$ we assign

$$\sum_{j:(i,j)\in E} \gamma_{ij} := \max(0, c_i + \sum_{j:(j,i)\in E} \gamma_{ji}). \tag{55}$$

Step 5. Finally we define a flow on the enriched graph $\tilde{G}$.

$$f_{ij} := \gamma_{ij}, \quad \forall (i,j) \in E, \tag{56}$$

$$f_{si} := c_i - \beta_i \quad \forall i \in V : c_i > 0, \tag{57}$$

$$f_{it} := -\sum_{j:(i,j)\in E} \gamma_{ij} + \sum_{j:(j,i)\in E} \gamma_{ji} \quad \forall i \in V : c_i < 0. \tag{58}$$

Let us check condition (49). For i: $c_i > 0$ we have

$$\begin{aligned} \sum_{j:(j,i)\in \tilde{E}} f_{ji} &= \sum_{j:(j,i)\in E} f_{ji} + f_{si} \overset{(57)}{=} \sum_{j:(j,i)\in E} \gamma_{ji} + c_i - \beta_i \\ &\overset{\text{Step 4}}{=} \sum_{j:(i,j)\in E} \gamma_{ij} = \sum_{j:(i,j)\in E} f_{ij} = \sum_{j:(i,j)\in \tilde{E}} f_{ij}; \end{aligned}$$

for i: $c_i \leq 0$ we obtain

$$\begin{aligned}\sum_{j:(i,j)\in\tilde{E}} f_{ij} &= \sum_{j:(i,j)\in E} f_{ij} + f_{it} \\ &\overset{(58)}{=} \sum_{j:(i,j)\in E} \gamma_{ij} - \sum_{j:(i,j)\in E} \gamma_{ij} + \sum_{j:(j,i)\in E} \gamma_{ji} \\ &= \sum_{j:(j,i)\in E} \gamma_{ji} = \sum_{j:(j,i)\in E} f_{ji} = \sum_{j:(j,i)\in\tilde{E}} f_{ji}.\end{aligned}$$

Constraint (50) is obvious since $\beta_i \geq 0$. To complete the proof we show constraint (51)

$$\begin{aligned}f_{it} &:= -\sum_{j:(i,j)\in E} \gamma_{ij} + \sum_{j:(j,i)\in E} \gamma_{ji} \\ &\overset{\text{Step 4}}{=} -\max(0, c_i + \sum_{j:(j,i)\in E} \gamma_{ji}) + \sum_{j:(j,i)\in E} \gamma_{ji} \\ &= \min(0, -c_i - \sum_{j:(j,i)\in E} \gamma_{ji}) + \sum_{j:(j,i)\in E} \gamma_{ji} \\ &= \min(\sum_{j:(j,i)\in E} \gamma_{ji}, -c_i) \leq -c_i.\end{aligned}$$

□

Theorem 7.14. *Let f be a solution of Maximum flow problem 7.12. Then one of the possible solutions of Linear program 7.11 can be expressed via:*

$$\beta_i = 0 \quad \forall i : c_i \leq 0, \tag{59}$$

$$\beta_i = c_i - f_{si} \quad \forall i : c_i > 0, \tag{60}$$

$$\gamma_{ij} = f_{ij} \quad \forall (i,j) \in E, \tag{61}$$

with the following optimal value of the objective function

$$Z^*(\beta, \gamma) = \sum_{i:c_i>0} c_i - F. \tag{62}$$

Proof. It is easy to see that (β, γ) defined by (59, 60, 61) satisfies conditions (46, 47, 48), i.e. it is a feasible solution of Linear program 7.11. What remains to be shown is that $\sum_i \beta_i$ is the smallest possible objective function value among all feasible solutions of Linear program 7.11. Assume there is a feasible solution $(\bar{\beta}, \bar{\gamma})$ to Linear program 7.11 with

$$\sum_i \bar{\beta}_i = Z^*(\bar{\beta}, \bar{\gamma}) < Z^*(\beta, \gamma) = \sum_i \beta_i. \tag{63}$$

Due to Lemma 7.13 there exists a flow $\bar{f} : \tilde{E} \to \mathbb{R}^+$, such that

$$\begin{aligned} F(\bar{f}) &\geq \sum_{i:c_i>0} c_i - \sum_{i:c_i>0} \bar{\beta}_i \\ &\overset{(63)}{>} \sum_{i:c_i>0} c_i - \sum_{i:c_i>0} \beta_i \\ &\overset{(59,60)}{=} \sum_{i:c_i>0} c_i - \sum_{i:c_i>0} c_i + \sum_{i:c_i>0} f_{si} \\ &= \sum_{i:c_i>0} f_{si} = F(f). \end{aligned}$$

We found $\bar{f}$ with a flow value $F(\bar{f}) > F(f)$, which contradicts the maximum flow expectations on f. Thus the assumption (63) was wrong, and this completes the proof. □

For the sake of completeness we summarize the results of this section in Algorithm 3. Finally we state the time costs of the latter algorithm in the following

Algorithm 3: Neyman-Pearson partitioning of a Layer

Input: Layer $L = P \cup N$, with P and N *positive* and *negative* counterparts.

Output: Partitioning $\{L^+, L^-\}$ if L is decomposable, reject otherwise.

1 Consolidate identical points of L into $V = \{v_i\}$ with multiplicity functions $p(v_i)$ and $n(v_i)$.
2 Construct graph G induced by $\leq$ within V.
3 Formulate Linear program 7.7 and find coefficients c_i of Linear program 7.10.
4 Formulate the dual Linear program 7.11.
5 Construct enriched graph $\tilde{G}$ with help of the constants c_i.
6 Solve Maximum flow problem 7.12.
7 If its flow value equals $\sum_{i:c_i>0} c_i$, then conclude: L is not decomposable, exit.
8 If its flow value less than $\sum_{i:c_i>0} c_i$, then use the complementary slackness conditions to find x_i of Linear program 7.10.
9 Use x_i to form sets $V^+ = \{v_i : x_i = 1\}$ and $V^- = \{v_i : x_i = 0\}$.
10 Assign the subsamples of the partitioning $L^+ = L \cap V^+$ and $L^- = L \cap V^-$ (recall, that unlike sets, samples may contain equal points multiple times).

Theorem 7.15. *Algorithm 3 complexity is at most* $O(|L|^3)$.

Proof. There are two sources of higher computational complexity in the algorithm:

- Construction of the graph G has a complexity of $O(|V|^2) \leq O(|L|^2)$, or with a transitive reduction $O(|V|^3) \leq O(|L|^3)$.
- Solution of Maximum flow problem 7.12 using e.g. [27] has a complexity of $O(|V|^3) \leq O(|L|^3)$.

Thereby both of them can be estimated from above by $O(|L|^3)$, and this finishes the proof. □

8 Numerical experiments

After finishing the theoretical considerations, it becomes necessary to validate the method experimentally. We do it using an artificially generated training and validation set. We do a prototype implementation of our trainable monotone combiner (TMC) approach and check if the procedure behaves as expected on two toy examples. Investigations, tests and adaptations on real data are issues for future work and lie beyond the scope of this paper.

For our experiments we use a MATLAB pattern recognition toolbox PRTools [33]. To be precise, all our sample data, base classifiers, and evaluations are based on the latest version PRTools5. We use a set of base classifiers to provide input features to the trainable monotone combiner, in the sense that each input feature is in its turn a discriminant function value of the corresponding base classifier. Consequently, one can expect a monotonic behaviour of the combined result, and therefore the trainable monotone combiner is applicable.

Our first example includes two base classifiers, and can be visualized by means of 2-D plots. The aim of the second example is to show, that an extension to n dimensions is easily possible. It contains three base classifiers; therefore, its resulting layers can not be plotted directly.

Example 8.1. We utilize the routine `gendatd` provided by PRTools5 to generate the data points of two *'difficult' normally distributed classes* in the 10-D space. As the first base classifier, we choose $k_1 =$ `qdc` trained on a sample of 64 randomly generated tuples, 32 per class. As the second, we take $k_2 =$ `ldc` trained on other 64 randomly generated tuples, 32 per class. `qdc` and `ldc` are the *quadratic* and *linear Bayes normal classifiers* correspondingly, provided by PRTools5. In order to specify training and validation data for the TMC combiner, we consider following data samples:

- C is a 10-D sample generated with `gendatd` that is transformed to the training sample, C_{comb}, for the TMC combiner via applying the base classifiers k_1 and k_2.
- D is a 10-D sample generated with `gendatd` containing 992 tuples that is transformed to the validation sample, D_{comb}, of the TMC combiner by means of (k_1, k_2).

The formal definition of the train and validation samples for the combiner reads:

$$\begin{aligned} C_{comb} &= \{(k_1(p), k_2(p)) \ \ \forall p \in C\}, \\ D_{comb} &= \{(k_1(p), k_2(p)) \ \ \forall p \in D\}. \end{aligned}$$

We are going to enlarge C additively starting at 32 tuples with a step of 64 until we arrive at 992 and observe the behavior of the approach as we increase the number of tuples $|C| = |C_{comb}|$ used to train the TMC combiner. The main results are summarized in Figure 12. The graph in Figure 12a shows how $|C|$ influences the ROC-error $= 1 - AUC$ on the training set C_{comb} (red line) and on the validation set D_{comb} (blue line). The most exciting and essential observation is, that the algorithm converges for each C_{comb} and the results seem to be meaningful, which verifies our theoretical considerations. What also seems interesting is that starting with a high enough $|C|$, further enlargement of the training set does not bring any significant advantages. In Figure 12b one can observe how the 'classical' classification error evolves according to PRTools5. The final training set is displayed in Figure 12c, and the resulting TMC layers can be seen in Figure 12d.

Example 8.2. This example differs from the first in two aspects: first, the base classifiers are now trained with 16 tuples per class. Secondly, we enrich the ensemble with an additional base classifier, `knnc` - the K-Nearest Neighbor Classifier provided by PRTools5. This increases the dimension of the intermediate feature space to three, which means we can plot neither training tuples, nor the TMC layers properly. The errors for this example are shown in Figure 13. They yield mainly similar behaviour as in the previous example.

We provide the MATLAB code of our implementation on `http://www.tmcombi.org` so that anyone can reproduce our tests without additional effort. It includes a reference implementation of the TMC combiner and the examples above.

9 Discussion: relationship to monotonic multi-class classification

In this section we would like to discuss how the approach of this paper relates to the known approaches of monotonic classification. The monotonic classification belongs to the field of ordinal classification, also known as ordinal regression, in which the considered classes have a natural order; see monograph [6] for a comprehensive overview. When determining the credit rating, the classes would be e.g. "AAA", "AA", "A", "BBB", "BB", "B", "CCC". The monotonic classification aims to determine a class so that the monotonicity conditions are met. In this case, this means that the class cannot decrease with increasing feature values in the input feature vector.

There are a variety of approaches dealing with the monotonic classification [34, 35, 36, 37]. We focus on those who have direct points of contact with our

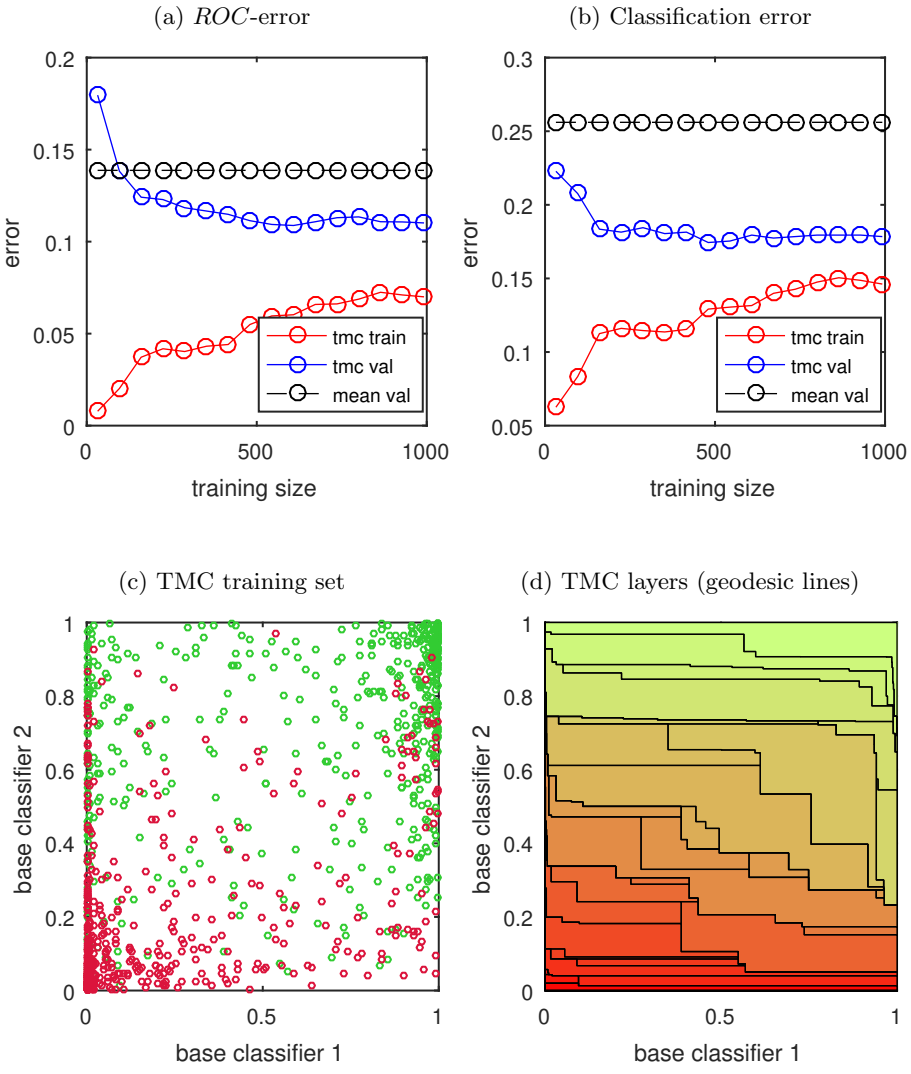


Figure 12: TMC combiner from Example 8.1.

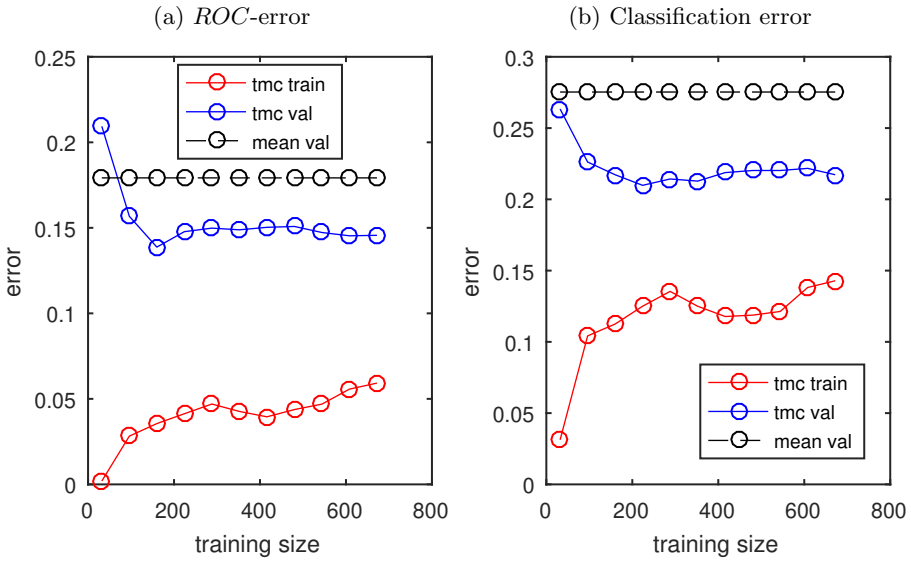


Figure 13: TMC combiner from Example 8.2.

approach. For this purpose, we represent the entire procedure described in this paper as a two-step process, where a well-proven approach is known for each of the steps.

- **Step 1**: Relabeling. In the context of monotonic classification, relabeling aims to change the class labels of the instances in order to minimize the number of monotonic violations in the training sample. This sounds very similar to what we do when creating the layers, with the difference that in our case, the layers are created dynamically and are not assigned each to a different class. The known relabeling approaches differ mainly in the function that is being optimized. For example, the approach based on isotonic regression [38] provides labeling that minimizes absolute or quadratic errors. The other series of approaches [39, 40, 35] set themselves the goal of minimizing the number of labels to be changed. Another approach [41] minimizes the deviations between the new and the original labels and the number of relabelings required.

- **Step 2**: Extension to the entire feature space. As soon as the monotonic violations have been eliminated, the classification provided on the labeled sample is extended to the entire feature space in order to be able to classify any arbitrary input. The ordered learning model [42, 1] can be used for this purpose, and we also follow the same lines in Section 3.

Seen in this way, our process follows the lines of the two known steps: relabeling and extension. However, there are two essential differences from the existing approaches:

1. We do not determine the number of layers in advance (in our case the layers play the role of classes from the approaches mentioned above). Instead, these are found automatically during training so that the objective function can achieve its optimum. This would correspond to a case of ordinal classification, in which the number of classes is optimally determined on the basis of the training sample.

2. We are optimizing a different objective function, namely the AUC, or equivalently the Neyman-Pearson ratios, which seems to be a very natural choice for the considerations we have described in this paper.

Finally, we would like to mention that this is not the first work related to monotonic classification, in which a linear program has appeared that can be reduced to a maximum flow problem. We refer to [43], in which the authors wrote down an isotonic classification problem as a linear program. Similar to our theory, they showed that the linear program can be reduced to a maximum flow problem. However, the two differences described above apply here as well.

In the rest of the section, we outline the ways in which TMC can be applied to monotonic classification with more than two classes. The following three approaches look promising. Their detailed investigations including benchmarking against known approaches are, however, outside the scope of this document and can be dealt with in the future. In the first case, we describe a situation in which training instances are labeled in two classes, but the classifier is expected to provide one of the several ordered classes as a result. The other two cases are based on a general case in which the training data is already divided into several classes.

Case 1: The training data is labeled in two classes. As an example of this, the case can be taken from the determination of the credit rating, where each feature vector within training data is labeled with "Default" (*negative* in notation of TMC) or "not Default" (*positive* in notation of TMC). We can directly trigger the TMC training for this binary labeled training sample and get a partitioning of the feature space into layers L_1, L_2, ... L_n. We then create clusters of successive layers and assign each such cluster to a class. Criteria for this clustering can vary from application to application. For example, approximately the same number of instances of the training sample can be expected per cluster with the intention that the classes will be populated similarly densely. In the case of credit rating, however, it makes sense to group the layers according to the value of the ratio of *negatives* to *positives* within the layer. In this context, this ratio is called the default rate. Thus, all layers with the default rate less than or equal to 0.0052 will fall into the "AAA" class (Moody's Corporate).

Case 2: Reduction of a k-class problem to $k-1$ binary problems. Now we are dealing with the generally formulated ordinal classification problem, where the training sample S is labeled into several classes. For each class c_i let subsample $S_i \subset S$ contain those feature vectors that are labeled to class c_i. In this way, the training sample can be represented as a union of differently labeled subsamples $S = \cup_{i=1}^{k} S_i$. We follow the approach of [44], Section 4.2 and train TMC $k-1$ times with *negatives* and *positives* chosen as follows:

TMC_1: $\quad$ *negatives* $= S_1$, $\quad$ *positives* $= S_2 \cup ... \cup S_k$,
TMC_2: $\quad$ *negatives* $= S_1 \cup S_2$, $\quad$ *positives* $= S_3 \cup ... \cup S_k$,
$\quad \ldots$
TMC_{k-1}: $\quad$ *negatives* $= S_1 \cup ... \cup S_{k-1}$, $\quad$ *positives* $= S_k$.

Let us now assume that x is an arbitrary input feature vector to be classified and $q_j(x) \in [0,1]$ its quality value provided by combiner $\text{TMC}_j, j = 1...k-1$. The confidence level for the class c_i for the input feature vector x will be defined as follows:

$$score(c_i, x) = \begin{cases} 1 - q_1(x), & i = 1, \\ q_{i-1}(x) - q_i(x), & 1 < i < k, \\ q_{k-1}, & i = k. \end{cases}$$

It can be shown that for every feature vector x, $\sum_{i=1}^{k} score(c_i, x) = 1$ and additionally for each class c_i, $0 \leq score(c_i, x) \leq 1$. The proof is omitted for space reasons.

Case 3: Reduction to a binary problem using weights. We assume that the classes are encoded by integers $c_i = i$, $i = 1..k$. For each feature vector x_j from the training sample S we denote its class label by $c(x_j)$. The idea is to create a binary problem by including a weighted *positive* and a weighted *negative* for each x_j in the new training sample of the auxiliary binary problem. The weights can be represented as functions:

$w_n(x_i)$ - the weight of the newly generated *negative* x_i, and
$w_p(x_i)$ - the weight of the newly generated *positive* x_i.

TMC is then applied to the training sample of *negatives* and *positives* generated in this way. It is worth noting that, due to its construction, TMC accepts arbitrary non-negative weights. Let us assume that the auxiliary binary problem is solved by means of TMC and the quality function $q(x)$ has been determined for a feature vector x to be classified. In order to complete the procedure, the value of the quality function $q(x)$ must be mapped onto a class label of the original multi-class problem using an appropriately selected reverse transformation $\tilde{c}(q(x)) \in [0, k]$.

We clarify our considerations based on a concrete example of the following weight functions

$$\begin{aligned} w_p(x_i) &= c(x_i), \\ w_n(x_i) &= 1 + k - c(x_i), \end{aligned}$$

and a reverse transformation function:

$$\tilde{c}(q(x)) = \frac{1+k}{1 + 1/q(x)}.$$

These functions are selected so that the class label for the instances in a perfectly monotonic training sample are kept within the trained TMC. We formulate this fact as a lemma.

Lemma 9.1. *If the training sample has no monotonic violations, and the quality function $q(x)$ is selected to be equal to the ratio of the number of positives to*

the number of negatives, then the class label determined by the TMC for each feature vector from the training sample equals to the label of this feature vector in the training sample, i.e. $\forall x_i \in S$, $\tilde{c}(q(x_i)) = c(x_i)$. *One speaks of monotonic violation if a larger feature vector of two comparable feature vectors is assigned to a smaller class.*

Proof. The proof arises from the fact that the optimal layer partitioning in this case contains exactly k layers that are directly assigned to the classes. □

It is worth noting that in general $c(q(x))$ does not necessarily need to provide an integer. Depending on the application, you can either round to an integer or accept a statement that a feature vector to be classified lies between two classes. Another important aspect is that the choice of $w_n(x_i)$ and $w_p(x_i)$ is crucial for the procedure, since the layers produced by the TMC depend essentially on the weights. This choice and its consequences must be examined in future work.

10 Discussion: open problems

At this point we would like to list the open questions that we did not address in the previous section. Some of the problems we state below are more theoretical, while others are more practical in nature. For some of these problems we provide ideas for further investigation; however, other problems are more complex.

1. **Application on real data.** In Section 8 we have checked the basic functionality of the TMC combiner on two toy examples. It would, however, be exciting to extend the tests to real world examples.

2. **Generalization to unseen data.** In this paper we focused on the minimization of training errors. Although a performance on unseen samples was out of the scope of this work, it must still be investigated. We would expect a result similar to the one from [44], Chapter 6. For an approach similar to ours, it was shown that for a classifier to be consistent it is sufficient to assume that the probability distribution with respect to the Lebesgue measure has a density in the feature space. We would expect that also in our case this guarantees that TMC converges against the best possible combiner as the size of the training sample increases.

3. **Complementary slackness conditions as a graph.** Step 8 of Algorithm 3 uses the complementary slackness conditions in order to convert the optimal solution of the dual formulation into the corresponding solution of the original Linear program 7.10. We are convinced that this step is equivalent to a certain graph problem; namely, the solution of the linear program can be determined by evaluating a *transitive closure* of a specially shaped graph. However, this claim needs to be rigorously examined, investigated, and proven, unless the corresponding result is already known but the author had not yet found it.

4. **Efficient computation of the discriminant function.** In Section 3 we discussed possibilities for optimization of the discriminant function evaluation. The optimal choice, however, still requires further investigation. We formalize the problem for those who wish to contribute. Suppose a finite set of pairwise incomparable points

$$T = \{t_i \in \mathbb{R}^k : \ t_i \not\leq t_j \ \& \ t_j \not\leq t_i \ \forall j \neq i\} \tag{64}$$

are given. Provide a data structure and a corresponding algorithm that for some $x \in [0,1]^k$ checks the following condition in the most efficient way:

$$\exists t_i \in T \text{ such that } t_i \leq x. \tag{65}$$

5. **Resource conservation.** Recall the resource conservation property from introductory Section 1. Suppose a threshold is set on a monotone combiner result. This means a separating polygon, depicted in Figure 2, is specified. It is clear that there are possibilities to avoid having to evaluate every input feature. The most efficient way to complete this, however, needs further investigation. We provide a formalization for potential contributors. Suppose a finite set of pairwise incomparable points defined by (64) and a sample $X \subset \mathbb{R}^k$ are given. Assume that an execution time of each feature can be measured and equals z_m, for $m = 1, \ldots, k$. Provide a way to save feature evaluations in such a way that the execution time of the combiner on X becomes as low as possible.

6. **Layer borders stabilization.** In Section 3 we defined a monotone combiner by means of its discriminant function: see Theorem 3.1 and Theorem 3.4. This discriminant function is equivalent to defining a set of separating polygons for $k = 2$, see Figure 3, and hyperplanes in case $k > 2$. The aim of these polygons or hyperplanes is to separate points belonging to adjacent layers L_i and L_{i+1}. The question at this point is whether it would be worth, and practically possible, to change this separation so that the resulting combiner becomes less sensitive to small training data perturbations. A separating polygon or a collection of hyperplanes lying exactly in between two layers would be a perfect choice. It would also bring a symmetry into the combiner, so that the classes become interchangeable. So,

   - can we construct and evaluate this separation efficiently?
   - If no, what other imperfect solutions can be considered?

7. **Feature selection.** There are a variety of feature selection methods that are suitable for monotonic classification [45, 46, 47, 48]. It would make sense to investigate the consequences of preselecting or filtering the features prior training the TMC.

8. **Best strategy of splitting and merging layers.** Within Algorithm 2 we are free to choose between splitting and merging the layers at each

iteration of Loop 1-6. Which strategy would yield faster convergence of Algorithm 2?

9. **Computational complexity I.** In Theorem 7.15 we showed an estimate from above for the complexity of Algorithm 3. Can it be improved?

10. **Computational complexity II.** Will a transitive reduction of the graph G prior to Algorithm 3 make sense? Will it if we consider Algorithm 2 in combination with Algorithm 3? Will this influence our choice of a maximum flow algorithm within Algorithm 3?

Acknowledgements

I thank my colleague Frank Pfeiffer for fruitful discussions. His comments influenced this work in many ways and his ideas were essential for the theory of Section 7. I would also like to thank my another colleague Rainer Lindwurm for proof reading of this manuscript and interesting discussions. Finally, I wish to thank my friends Gennadiy Averkov, Jeka Volfson and Stas Lyakhov for their useful advices.

References

[1] Arie Ben-David, Leon Sterling, and Yoh-Han Pao. Learning and classification of monotonic ordinal concepts. *Computational Intelligence*, 5(1):45–49, 1989.

[2] Chris Carter and Jason Catlett. Assessing credit card applications using machine learning. *IEEE expert*, 2(3):71–79, 1987.

[3] Daniel A Bloch and Bernard W Silverman. Monotone discriminant functions and their applications in rheumatology. *Journal of the American Statistical Association*, 92(437):144–153, 1997.

[4] Patrick Royston. A useful monotonic non-linear model with applications in medicine and epidemiology. *Statistics in medicine*, 19(15):2053–2066, 2000.

[5] David Gamarnik. Efficient learning of monotone concepts via quadratic optimization. In *Proceedings of the eleventh annual conference on computational learning theory*, pages 134–143. ACM, 1998.

[6] José-Ramón Cano, Pedro Antonio Gutiérrez, Bartosz Krawczyk, Michał Woźniak, and Salvador García. Monotonic classification: An overview on algorithms, performance measures and data sets. *Neurocomputing*, 341:168–182, 2019.

[7] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[8] Sergey Tulyakov, Stefan Jaeger, Venu Govindaraju, and David Doermann. Review of classifier combination methods. In *Machine Learning in Document Analysis and Recognition*, pages 361–386. Springer, 2008.

[9] J. Neyman and E. S. Pearson. On the Problem of the Most Efficient Tests of Statistical Hypotheses. *Philosophical Transactions of the Royal Society of London Series A*, 231:289–337, 1933.

[10] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[11] Lei Xu, Adam Krzyżak, and Ching Y Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3):418–435, 1992.

[12] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[13] Josef Kittler and Mohamad Hatef. Improving recognition rates by classifier combination. In *In Proc. 5th Int. Workshop on Frontiers of Handwriting Recognition*, pages 81—-102, 1996.

[14] Josef Kittler, Mohamad Hatef, Robert PW Duin, and Jiri Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, 1998.

[15] Giorgio Fumera and Fabio Roli. Performance analysis and comparison of linear combiners for classifier fusion. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 424–432. Springer, 2002.

[16] Luís A Alexandre, Aurélio C Campilho, and Mohamed Kamel. Combining independent and unbiased classifiers using weighted average. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 495–498. IEEE, 2000.

[17] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[18] Diane Greene. An implementation and performance analysis of spatial data access methods. In *Data Engineering, 1989. Proceedings. Fifth International Conference on*, pages 606–615. IEEE, 1989.

[19] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, SIGMOD '90, pages 322–331, New York, NY, USA, 1990. ACM.

[20] Scott T Leutenegger, Mario A Lopez, and Jeffrey Edgington. Str: A simple and efficient algorithm for r-tree packing. In *Data Engineering, 1997. Proceedings. 13th International Conference on*, pages 497–506. IEEE, 1997.

[21] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. In *Proceedings of the second international conference on Information and knowledge management*, pages 490–499. ACM, 1993.

[22] Rainer E Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems, Revised Reprint*. Siam, 2009.

[23] Rob Potharst and Adrianus Johannes Feelders. Classification trees for problems with monotonicity constraints. *ACM SIGKDD Explorations Newsletter*, 4(1):1–10, 2002.

[24] Jerzy Neyman and Egon S Pearson. *On the problem of the most efficient tests of statistical hypotheses*. Springer, 1992.

[25] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[26] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[27] Vishv M. Malhotra, M Pramodh Kumar, and Shachindra N Maheshwari. An $O(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7(6):277–278, 1978.

[28] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[29] James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774. ACM, 2013.

[30] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[31] R. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2007.

[32] Robert G Bland. New finite pivoting rules for the simplex method. *Mathematics of operations Research*, 2(2):103–107, 1977.

[33] R.P.W. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, D.M.J. Tax, and S. Verzakov. *Prtools, a matlab toolbox for pattern recognition*, 2004.

[34] Arie Ben-David. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19(1):29–43, 1995.

[35] Wouter Duivesteijn and Ad Feelders. Nearest neighbour classification with monotonicity constraints. *Machine Learning and Knowledge Discovery in Databases*, pages 301–316, 2008.

[36] Nicola Barile and Ad Feelders. Nonparametric monotone classification with moca. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 731–736. IEEE, 2008.

[37] Wojciech Kotłowski and Roman Słowiński. Rule learning with monotonicity constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 537–544. ACM, 2009.

[38] Richard Dykstra, John Hewett, and Tim Robertson. Nonparametric, isotonic discriminant procedures. *Biometrika*, 86(2):429–438, 1999.

[39] AJ Feelders, Marina Velikova, and Hennie Daniels. Two polynomial algorithms for relabeling non-monotone data, 2006.

[40] Michaël Rademaker, Bernard De Baets, and Hans De Meyer. Loss optimal monotone relabeling of noisy multi-criteria data sets. *Information Sciences*, 179(24):4089–4096, 2009.

[41] Wim Pijls and Rob Potharst. Repairing non-monotone ordinal data sets by changing class labels. Tech. rep., Econometric Institute, Erasmus University Rotterdam, 2014.

[42] Arie Ben-David. Automatic generation of symbolic multiattribute ordinal knowledge-based dsss: methodology and applications. *Decision Sciences*, 23(6):1357–1372, 1992.

[43] Ramaswamy Chandrasekaran, Young U Ryu, Varghese S Jacob, and Sungchul Hong. Isotonic separation. *INFORMS Journal on Computing*, 17(4):462–474, 2005.

[44] Wojciech Kotlowski and Roman Slowinski. On nonparametric ordinal classification with monotonicity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(11):2576–2589, 2012.

[45] Qinghua Hu, Weiwei Pan, Lei Zhang, David Zhang, Yanping Song, Maozu Guo, and Daren Yu. Feature selection for monotonic classification. *IEEE Transactions on Fuzzy Systems*, 20(1):69–81, 2012.

[46] Qinghua Hu, Weiwei Pan, Yanping Song, and Daren Yu. Large-margin feature selection for monotonic classification. *Knowledge-Based Systems*, 31:8–18, 2012.

[47] Weiwei Pan, Qinghua Hu, Yanping Song, and Daren Yu. Feature selection for monotonic classification via maximizing monotonic dependency. *International Journal of Computational Intelligence Systems*, 7(3):543–555, 2014.

[48] Weiwei Pan and Qinhua Hu. An improved feature selection algorithm for ordinal classification. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 99(12):2266–2274, 2016.