TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

DISSERTATION

Cost Sensitive Screening Methods for Binary Classification

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.-Prof. Dipl.-Ing. Dr. techn. Peter Filzmoser, Institut für Stochastik und Wirtschaftsmathematik (E105)

eingereicht an der Technischen Universität Wien an der Fakultät für Mathematik and Geoinformation

von

Dipl. Ing. Fabian Schroeder M.Sc.
Matrikelnummer 00300500

Diese Dissertation haben begutachtet:

Prof. Priv. Doz. DI Dr. Florian Frommlet	Univ.-Prof. Mag. Dr. Andreas Futschik

Wien, 17. Mai 2018

Dipl. Ing. Fabian Schroeder M.Sc.

Erklärung zur Verfassung der Arbeit

Dipl. Ing. Fabian Schroeder M.Sc.
Starkfriedgasse 31,1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 17. Mai 2018

Dipl. Ing. Fabian Schroeder M.Sc.

Acknowledgements

I would like to express my sincere gratitude to my advisor Professor Peter Filzmoser for his continuous support on this long dusty path called PhD. His academic as well as personal guidance is much appreciated. Furthermore, I would like to thank my supervisor at the AIT Klemens Vierlinger for lively discussions which sparked the following ideas and for the space he left me to pursue them.

I am forever indebted to my family, in particular my mother Renée for standing as a role model in many regards. Above all, I would like to thank Marlies, mother of my children, Sophia, Valerie, and Moritz. Family happiness has been my tower of strength throughout these exciting years.

Abstract

In high-dimensional classification tasks, e.g., case-control experiments for biomarker detection, variable filters constitute a simple and scalable method to discard uninformative variables. When screening for interesting variables the neglect of the operating conditions of the classification task can lead to false conclusions. This thesis, thus, proposes filtering statistics based on the expected prediction error of a univariate classifier. The choice of classifier will generally determine the characteristics of the filtering method. An obvious parametric approach was the Bayesian classifier for a mixture of Gaussian class conditionals. This approach, however, relies heavily on the parametric assumptions and any deviation from these will reduce the performance of the filter dramatically. Opting for a non-parametric classifier seems reasonable in the context of screening thousands of different variables. Thus, instead of assuming a parametric family for the class conditionals we will assume that the optimal classifier is a member of the family of threshold or interval classifiers. This assumption should hold true for a great number of different distributions. Furthermore, these methods exhibit another interesting property. It is possible to obtain the exact finite sample distribution of the test statistic under the null hypothesis of equal class conditional distributions by means of a fast recursive algorithm. In this thesis, I have studied the proposed screening methods analytically and evaluated their screening characteristics by means of simulated as well as real data.

Kurzfassung

In hochdimensionalen Klassifikationsaufgaben, z.B. in Fall-Kontroll-Studien für die Detektion von Biomarkern, sind Variablenfilter eine einfache und skalierbare Methode uninformative Variablen zu verwerfen. Die Nichtberücksichtigung der Operation Conditions bei der Suche nach interessanten Variablen kann jedoch zu falschen Schlüssen führen. Daher untersuche ich in dieser Arbeit die Verwendung von Variablenfiltern, die auf dem erwarteten Prognosefehler von eindimensionalen Klassifikatoren basieren. Grundsätzlich dürfte die Wahl des Klassifikators die Eigenschaften des daraus resultierenden Filters bestimmen. Ein naheliegender parametrischer Ansatz wäre der Bayes Klassifikator unter der Annahme von normalverteilten Klassen. Dieser Filter ist jedoch stark abhängig von der Verteilungsannahme und schneidet schlecht ab wenn die Daten von der Annahme abweichen. Ein nichtparametrischer Ansatz wäre daher für die Vorauswahl von hunderttausenden Variablen verschiedenem Ursprungs zu bevorzugen. Statt einer Annahme über die Verteilung der Klassen, könnte man einfach annehmen, dass der optimale Klassifikator von einer bestimmten Gestalt ist, z.B. dass der Annahmebereich für eine Klasse ein (möglicherweise unbeschränktes) Intervall ist. Diese Annahme dürfte für eine große Anzahl von Verteilungsfamilien erfüllt sein. Ausserdem haben die daraus resultierenden Methoden eine weitere interessante Eigenschaft: es ist möglich die Verteilung der Teststatistik unter der Nullhypothese für endliche Stichproben exakt zu bestimmen. Die Berechnung basiert auf einem schnellen rekursiven Algorithmus. In dieser Arbeit untersuche ich die vorgeschlagenen Filtermethoden analytisch sowie mit Hilfe von simulierten und echten Daten.

Foreword

In the last two decades, we have witnessed the advent of modern high-throughput devices. These were made possible by a myriad of different technologies in diverse fields such as robotics, sensor technologies but also assay systems and informatics. It has become possible to measure hundreds of thousands of different entities at dramatically lower cost. These advancements are reshaping the research and development pathways for drugs, vaccines, and diagnostics and have led to the emergence of a completely novel kind of experiment: the high-throughput screening experiment.

During my employment at the Department of Molecular Diagnostics at the Austrian Institute of Technology, I have contributed to the design and the evaluation of many such studies. The objective was always the same: the development of diagnostic models. In this context, the screening experiments are most often referred to as biomarker detection. A biomarker, or biological marker, refers to a "characteristic that is objectively measured and evaluated as an indicator of normal biological processes, pathogenic processes, or pharmacological responses to a therapeutic intervention", see Group (2001). Genome-wide biomarker discovery, a method comprising the systematic screening of potentially hundreds of thousands or even millions of different entities, can aid the researcher in many different ways. Among other objectives, it allows the researcher to identify those entities which play a crucial role in the development of a disease, to gain a mechanistic understanding of the differences, or to develop surrogate endpoints for clinical outcomes which cannot be observed.

The typical setup for such an experiment is a case-control or treatment-control study, for an in-depth introduction, see Schlesselman and Stolley (1982). This is a type of observational study, which examines the difference between the case and the control group with respect to the measured entity. As compared to a randomized controlled trial they "require fewer resources but also provide less evidence for causal inference". Typical clinical endpoints for the case and control groups include diseased vs. non-diseased, malign vs. benign tumor or treatment vs. placebo. Without any prior knowledge, thousands sometimes even millions of different potential biomarkers are measured. This completely naive approach is suitable for exploring the variables and for formulating new hypotheses. Most importantly, its main purpose is to identify those variables which show a significant discriminatory power between the case and the control groups. The selected biomarkers are then validated in the second round of experiments using new samples and a different platform.

The dimensionality of a dataset from high-throughput technology was unprecedented and has, thus, incited a lot of statistical research, mainly in the field of model selection. This is a common theme in statistics that technological advances in other scientific fields spur statistical research and lead to completely new questions. When I first looked into the literature on high-dimensional classification, I was overwhelmed by the amount of research that had been conducted in such a short time span. But I was also surprised by the fact that there were only a few available methods for filtering. In particular, there were no available methods, which allowed to rank and select the variables taking the operating characteristics of the classification task into account. The so-called operating conditions describe the characteristics of the classification task and include the misclassification costs and the class distribution in the population. The diagnosis of patients is a classification task which is usually characterized by a highly unbalanced population and strongly asymmetric costs. Secondly, it was surprising that univariate classification was not used as a variable filter, which appeared most obvious. This seemingly easy approach was rarely chosen and, thus, little experience was available which could be built upon. At some point in time, it became obvious that this topic would deliver sufficient content for my PhD thesis.

Contents

CHAPTER 1

Introduction

The class of statistical models that is suitable when the response is discrete - as in case-control experiments - are referred to as classification models, see Hastie et al. (2001) for an introduction. Data from screening experiments, however, have a number of important idiosyncrasies, which render many classifiers unsuitable.

Firstly, the extremely high number of measured entities, a vast majority of which are irrelevant for the classification task or redundant. If the number of variables is greatly superior to the number of instances ($p \gg n$), a number of problems occur, which are subsumed by the term *curse of dimensionality.*

Secondly, the purpose of the screening experiment. Whereas most classification tasks aim at identifying a single classifier or statistical model with an optimal prediction performance, the most important final outcome of a screening experiment is usually a subset of variables. These are subsequently subject to follow-up experiments with possibly more complex experimental settings and/or more samples. This experiment is usually termed *validation experiment* and is essential due to the high number of incorrectly identified variables.

Idiosyncrasy number three concerns the medical context of screening experiments. Especially for the purpose of diagnosis, a false positive has different consequences than a false negative and the positives are relatively rare in the population. Thus, classifiers are applied which can account for the *operating conditions* (OCs) of the classification task. The OCs comprise the misclassification costs and the prevalence of the classes in the population, see Viaene and Dedene (2005) for an overview. Medical diagnosis is a prototypical example of a classification task where asymmetric misclassification costs or a low prevalence are the rule rather than the exception. This is due to the fact that most diseases are relatively rare and the inability to detect a diseased patient has more severe consequences than misdiagnosing a healthy one. Disregarding the OCs might lead to the selection of suboptimal variables. Nevertheless, it is still often ignored that the assessment of a classifier by means of the error rate, or the accuracy, in such situations is flawed, see e.g. Provost and Fawcett (2001) or Provost et al. (1998). The OCs subsume the conditions under which a classifier is applied, see e.g. Hernández-Orallo et al. (2012). Numerous possibilities to account for asymmetric costs or unbalanced classes in a population have been proposed, for an overview see e.g. Viaene and Dedene (2005).

While it is standard for clinical studies to evaluate the classifier by its sensitivity and its specificity, no OC sensitive methods exist for filtering variables. Also, the benefit of OC sensitive filtering on the performance of the subsequent classifier has not been subject to studies, yet. Cost sensitive classification problems arise not only in the medical field but also in diverse fields such as agricultural product inspection, credit card fraud, industrial production processes, text classification, etc.

1.1 Model Selection and Filtering

There are two main approaches to dimension reduction: feature extraction and feature selection. The first attempts to derive new variables from the existing ones, usually by finding interesting linear combinations, e.g. Principal Component Analysis or Independent Component Analysis. While this approach often builds models with a good predictive power, the interpretability is generally lost or difficult at least. They are, thus, unsuitable, for the purpose of filtering. The latter approach simply selects a subset of the variables. Ideally, variable selection methods search the entire space of possible subsets, however, as the number of features p increases, searching among all 2^p competing candidate subsets quickly becomes infeasible.

Following the categorization of Kohavi and John (1997), feature selection methods can be categorized into filters, wrappers, and embedded methods. See Guyon and Elisseeff (2003) or Dash and Liu (1997) for a general introduction.

One of the first attempts to tackle the $p \gg n$ problem was the LASSO (least absolute shrinkage and selection operator), which basically introduces a penalty term to the problem of minimizing the residual sums of squares punishing the number of variables included in the model, see Tibshirani (1996). Embedded model selection can be characterized by the fact that the model selection is inherent to the method. In most cases, this is implemented by means of a two-part objective function. The first term represents the goodness-of-fit of the model, the second introduces a penalty term that punishes the number of variables of the model. Another important variant is the elastic net (Zou and Hastie, 2005). These methods can be distinguished by their modifications of the penalty function. While embedded variable selection methods show a good prediction performance, they tend to overestimate the number of variables. Furthermore, they allow to gain little insight into the set of variables and, thus, they constitute a black box to the domain specialist.

Wrapper methods choose an altogether different approach. They assess subsets of variables according to their usefulness to a given predictor. In order to construct a wrapper method one needs to define 1) how to assess the prediction performance of the classifier and 2) how to search for an optimal subset of variables. There are several approaches. The best subset selection method evaluates all possible combinations of the p different variables. This, however, requires the assessment of 2^p different models and is only an option for sample spaces with 25 variables at most since $2^{25} \sim 3 * 10^7$. Other search strategies include forward selection or backward elimination (Kittler, 1978), simulated annealing, genetic algorithms (Holland, 1992). The downside to this approach is the computational cost, which can be prohibitive at times.

Variable selection by means of a filter consists of two steps. First, one needs to define a function according to which the variables are ranked and subsequently one needs to decide on how many variables to select. Filters fall into three different classes based on the criteria which they are based on: correlation, mutual information, and single variable classifiers. Popular members include the Fisher Score (Duda et al., 2012), the t-test (Student, 1908; Welch, 1947), which are based on the correlation between the variable and the class labels, or methods based on mutual information, see e.g. Peng et al. (2005). Interestingly, univariate classifiers, that model the probability of belonging to a given class for every point in the domain, are rarely used for filtering. This is surprising since they exhibit considerable advantages for filtering, e.g. the possibility to take operating conditions into account.

Filters are simple to use and easy to interpret. Computationally, they are efficient since they only require computing p scores and sorting them. Thus, for many different tasks filtering is the only possible approach. Even the consideration of pairs of variables is infeasible for most high-throughput screening experiments since for a sample space of 10^5 different variables this would require 10^{10} different models to be evaluated. Statistically, filtering is less prone to overfitting since it introduces a bias but it may have considerably less variance than other methods, see Hastie et al. (2001).

Filters are not necessarily used to build predictors since often top ranking variables are highly redundant. In this light, filtering is often considered a pre-processing step. Many variable selection algorithms include filtering as an auxiliary selection method, yielding a dramatic reduction of the dimension of the sample space before other methods are applied. It is, thus, often used as a first and crude step in a multi-step model selection procedure. Filters can, however, also be seen as an exploratory step. One can learn about the number of variables with a discriminatory power and one can analyse the correlations among them. The domain specialist is often capable of interpreting the results.

While all filters are able to rank variables, only those that are formulated as a statistical test can decide how many variables to select. However, in most experiments, the number of variables that will be selected is decided upon based on other considerations but statistical ones, e.g. budget constraints or technical constraint such as the number of possible variables on the array.

This approach is, however, far from optimal. A variable does not need to exhibit a strong univariate predictive power to improve the performance of a multivariate classifier. In fact, it is even possible to build a perfect classifier with two variables that have no predictive power at all.

1.2 Location Tests

One of the most widely used methods for univariate filtering is a location test, in particular, the t-test. Locations tests infer whether there is a significant difference in the central location of two classes. The t-test (Student, 1908) or the Welch test (Welch, 1947), a modification for the situation when the variances of the class distributions are unequal, were definitively never intended to serve as a variable filter, yet have become a standard method for this purpose, see e.g. Jaeger et al. (2003) or Su et al. (2003) for microarray data and Wu et al. (2003) or Levner (2005) for mass-spectrometry data. "Student's t-statistic is finding applications today that were never envisaged when it was introduced more than a century ago" (Delaigle et al., 2011). There are even modifications of the t-test just for this purpose, e.g. the moderated t-test (Smyth et al., 2004).

One can only speculate about the reasons for its popularity. Delaigle et al. (2011) explicitly mention "its robustness against heavy-tailed sampling distributions". By virtue of the Central Limit Theorem, the Studentized mean's limiting distribution is standard normal. The only requirement is finite variances. However, the result holds even for relaxed assumptions (Giné et al., 1997). Thus, the distribution of the studentized mean is asymptotically normal for a vast family of distributions and so is the t-statistic, as the difference of two studentized means. However, we would question the robustness of the t-test, although Delaigle et al. (2011) argue otherwise. The breakdown point of the t-test is $1/n$ since by increasing the value of a single observation we can arbitrarily increase the test statistic. This issue will be resolved in a simulation experiment.

Furthermore, there are not many other popular filters which are formulated as a statistical test. The limiting distribution of the test statistic under the null hypothesis is known and it is, thus, possible to obtain a p-value for every variable. This p-value can subsequently be adjusted for multiple testing. The t-test as a filtering method, thus, does not only rank the variables but it also answers the question, how many variables to select.

In the following let us illustrate a few shortcomings of the t-test as a filter statistic by means of a simple example. These shortcomings were the starting point for this thesis. Consider operating conditions characterized by asymmetric costs ($c_0 : c_1 = 1 : 3$) and balanced classes ($\pi_0 = \pi_1 = 0.5$). Under these circumstances we want to select a variable which best discriminates between classes among the three depicted in the left column of Figure 1.1. The variables $Z_i = (X_i, Y_i)$ for $i = 1, 2, 3$ have Normal distributions for each class $(X_i|Y_i = y) \sim N(\mu_{i,y}, \sigma^2_{i,y})$ with equal means but unequal variances

$$\begin{array}{lllll} Z_1: & \mu_{1,0} = -0.5, & \mu_{1,1} = 0.5, & \sigma^2_{1,0} = \sqrt{1/4}, & \sigma^2_{1,1} = \sqrt{7/4}, \\ Z_2: & \mu_{2,0} = -0.5, & \mu_{2,1} = 0.5, & \sigma^2_{2,0} = 1, & \sigma^2_{2,1} = 1, \\ Z_3: & \mu_{3,0} = -0.5, & \mu_{3,1} = 0.5, & \sigma^2_{3,0} = \sqrt{7/4}, & \sigma^2_{3,1} = \sqrt{1/4}. \end{array}$$

If one had to predict the unobserved class membership based on the value of X, which of the three variables would yield the best results? If we based our decision on the test statistic of the t-test, we would be indifferent between the three. The Welch Test statistic, a variant of the t-test for unequal variances, is defined as

$$T = \frac{\hat{\mu}_0 - \hat{\mu}_1}{\sqrt{\frac{\hat{\sigma}_0^2}{n} + \frac{\hat{\sigma}_1^2}{m}}},$$

where $\hat{\mu}$ and $\hat{\sigma}$ denote the maximum likelihood estimators of μ and σ, respectively. n and m are the training set sample sizes of the negative class and the positive class, respectively. The distribution of T under the null hypothesis $H_0 : \mu_0 = \mu_1$ can be approximated by a t-statistic with ν degrees of freedom where ν is a function of $\hat{\sigma}_0^2, \hat{\sigma}_1^2, n$ and m, see Welch (1947).

It is easy to verify that the Welch Statistic for Z_1, Z_2, and Z_3 have the same expected value. However, the expected prediction errors, defined in (2.4), that can be obtained by using a univariate classifier on the respective variables differ substantially. In order to compare the expected prediction error of a variable the applied classifier must not be arbitrary. Thus, we used the so-called Bayes classifier, which is defined by its characteristic that it minimizes the expected prediction error, to compare them. Consider the ROC curves of the Bayes classifiers of Z_1, Z_2 and Z_3 in the middle column of Figure 1.1. ROC curves plot the false positive rate against the true positive rate and give a nice graphical impression of the performance of a binary classifier, see e.g. Bradley (1997). The dashed grey lines are the isocost lines, indicating all points in the ROC space that incur an equal prediction error. By definition one is indifferent between all points indicated by these lines. The slopes of the isocost lines are characterized by the asymmetric cost, e.g. we are prepared to exchange three false positives for one false negative. The higher the isocost lines, the lower the induced prediction error, the better. Thus, Z_3 performs much better than Z_1 under these operating conditions. In the confusion matrices in the right column, we can see that Z_3 not only yields a lower false negative rate compared to Z_2 but that the false positive rate is also lower. The expected prediction error of the Bayes classifier of Z_1, Z_2, and Z_3 is 0.468, 0.445, and 0.27, respectively.

This simple example illustrates that common filter statistics, such as the t-test, completely ignore the asymmetric structure in the variances of these variables. In fact, this is the case for all location tests. We will see later on that a strong inequality between the variances, an important source of information about the class membership, reduces the power of the t-test.

In the simulation studies in Section 6 we will see that even small deviations from the assumption of Gaussian class conditional distributions, e.g. a contamination with outliers or skewness, can strongly affect the performance. In the real data examples, we will see that strong outliers strongly affect the test statistic and, thus, the conclusions that can be drawn from them.

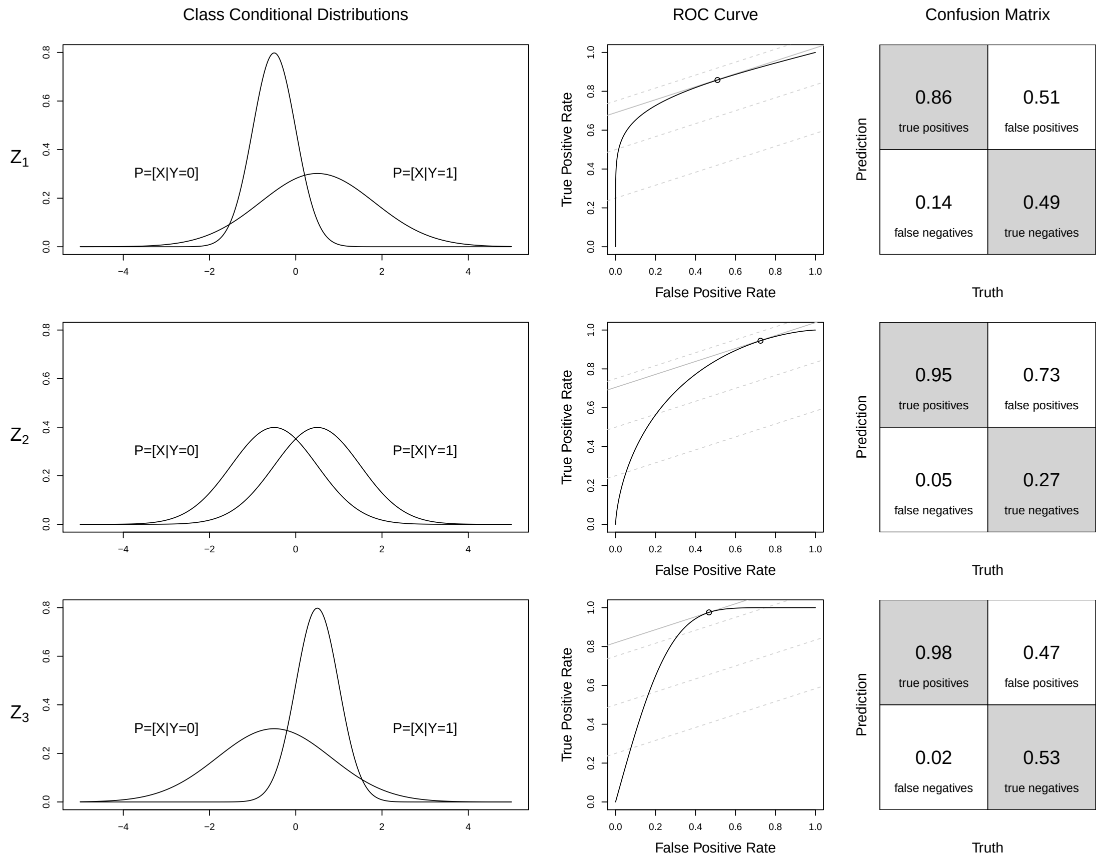


Figure 1.1: The class conditional distributions (left column) and the corresponding ROC curves of the Bayes Classifier of the random variables of Z_1, Z_2, and Z_3 (center column). The isocost lines in grey indicate the set of points in the ROC space which yield the same expected cost given a cost ratio of $c_1 : c_0 = 3 : 1$. The confusion matrices are depicted in the right column. The t-statistic of all three variables is identical, however, the expected prediction error of the Bayes classifier of Z_1, Z_2, and Z_3 is 0.468, 0.445, and 0.27, respectively.

Nonparametric location tests

Nonparametric location tests are inexact in the sense that the exact finite sample distribution of the test statistic under the null hypothesis, also referred to as null distribution (ND), can rarely be obtained. A notable exception is the Wilcoxon Rank-Sum Test (Wilcoxon, 1945), where exact inference is possible up to a sample size of about 30. Most methods rely on asymptotic results, which is critical for small sample sizes and when the p-values of the test statistics used for filtering variables are corrected for multiple testing. Small errors in the calculation of the p-values can be amplified by the adjustment.

The most common nonparametric approach is the permutation test, also known as randomization test (Edgington, 2011). It establishes the ND by calculating all possible values of the test statistic under rearrangements of the class labels. However, even for small sample sizes, an exhaustive calculation is computationally infeasible. For example, two different class labels with 30 samples each can be permuted in $\binom{60}{30} > 10^{17}$ different ways. Thus, one resolves to draw permutations randomly. This approach is problematic because the p-values of the most promising variables will lie on the tail of the ND. Since only a few random permutations will yield values in this region of the ND, the estimation will be most inaccurate. Knijnenburg et al. (2009) suggest to approximate the tail of the ND by curve fitting techniques. Thus, although permutation tests are sometimes referred to as exact tests, in practice exact p-values cannot be obtained.

1.3 Multiple Testing

Many filters are formulated as statistical tests. Due to the high number of conducted tests in most screening experiments, filtering is generally conceived as a multiple testing problem, see Miller (1981), Hsu (1996), or Pigeot (2000) for an in-depth introduction. Leaving the p-values unadjusted would yield approximately αp false positive variables, where α is the significance of the test. The correction of the p-value for the purpose of reducing the number of false positives comes at the cost of increasing the probability of producing false negatives, i.e., reducing statistical power. The decision on how to correct the p-values is, thus, a delicate task and depends mainly on the purpose of the screening experiment. The oldest approach is to control the *familywise error rate* and is attributed to Carlo Bonferroni. The Bonferroni correction is, however, too conservative for a large number of tests. Thus, recently a number of alternatives have been suggested, e.g *the per family error*, Tukey (1953) or the *false discovery rate* Benjamini and Hochberg (1995).

1.4 Outline

In this thesis, we will study the use of univariate classifiers as variable filters. In order to incorporate the operating conditions of the classification task, the filter statistic will consist of the estimated prediction error (EPE) of the classifiers. In Section 2 all the necessary terms are defined and we will establish that the EPE can be denoted as a weighted mean of the expected false positive and false negative rates.

The first univariate classifier under consideration is the Bayes Classifier for Gaussian class conditional distributions, see Section 3. If the classes are known to follow a Gaussian distribution with known parameters, it is possible to deduce the classifier that minimizes the EPE. If the parameters are, however, unknown they can be estimated from the sample, e.g. by means of the maximum likelihood estimator. This approach is equivalent to the *Quadratic Discriminant Analysis* (QDA) or the *Linear Discriminant Analysis* (LDA) under the assumption that the Gaussian class conditionals have equal variances. Thus, the first filter statistic that will be introduced will be referred to as EBC. It is essentially the prediction error of the Bayes classifier for Gaussian class conditionals and will serve as a benchmark for extensive simulation studies in Section 6 since we can simulate data for which it constitutes the optimal filter.

In Section 3 we will derive expressions for EBC and show that the univariate LDA sets a threshold t on the sample space and maps all instances left of this threshold to one class and all instances to the right of the threshold to the other class. The threshold can be written as a function of the estimated parameters of the Gaussian distributions. The QDA maps all instances within an interval to one class and all instances in its complement to the other. Again, the boundaries of the interval can be written as a function of the estimated parameters of the Gaussian class conditional distributions.

It is reasonable to assume that the set of distributions that yields a threshold or interval classifier as their respective Bayes classifier is vast. In particular, many distributions with a kurtosis or skewness which differs from the Gaussian family will be among them. Thus, Sections 4 and 5 will introduce non-parametric counterparts to these two filter statistics, ETC, the *Expected Prediction Error of the Threshold Classifier*, and EIC, the *Expected Prediction Error of the Interval Classifier*. Instead of making a parametric assumption about the data, we will assume that the optimal classifier is a member of a family of classifiers, and calculate the minimal EPE among all members of those respective families.

Another important question concerns the significance of the obtained estimates. Calculating the p-value would allow us to use the test statistics not only to rank variables but also to decide on how many variables to select based on the data. For this purpose, we would need to derive the distribution of EBC, ETC and EIC under the assumption that the continuous random variable exhibits no information about the class. More formally, the null hypothesis can be characterized by means of the class-conditional distributions (CCDs) of the random variable. If the CCDs differ from one another, predicting the class membership based on the value of the continuous random variable is superior to random guessing. Sections 4.3 and 5.3 will derive the null distributions for finite samples and a set of operating characteristics by means of a fast recursive algorithm.

Section 6 will shed light on the capacity of the introduced filter statistics to select signal variables out of a large number of noise variables. The simulation scenarios A to D were designed to analyze the power as well as the robustness to outliers and skewness. The main question of the simulation study E and the real data studies concerns the importance of cost-sensitive filtering for the purpose of building a cost-sensitive classifier. In other words, if a filter is used to reduce the dimension of the sample space and subsequently a classification model is built with the reduced set of variables, what is the benefit if the filter is OC sensitive?

CHAPTER 2

Preliminaries

The purpose of *supervised classification* is to conclude on the discrete state or class of an instance by observing one or many variables which we expect to contain information about the state. Think about assigning a patient with an unknown health status to one of two classes, healthy or ill, based on a set of potential biomarkers $\mathcal{X} = (X_1, \dots, X_p)$.

More formally, consider the random variable (X, Y) defined on $\mathcal{X} \times \mathcal{Y}$, where X is real valued ($\mathcal{X} \subseteq \mathbb{R}^p$) and Y is discrete representing the classes of the instance. Let P denote their joint probability distribution. Let the random variables X conditional on a certain value of Y be denoted $X^i := (X|Y = i)$ and its respective distributions F^i.

The purpose of supervised classification is to predict the class labels Y based on the realization of X. This is the task of a classifier.

Definition 1 *A* classifier $\delta : \mathcal{X} \to \mathcal{Y}$ *is a mapping that attaches a class membership to a realization of* X*. A classifier is called* binary *if* $|\mathcal{Y}| = 2$*. If* $\mathcal{X} \subseteq \mathbb{R}$ *we will speak of a univariate classifier.*

Many classifiers allow the representation as a model and a threshold. This distinction allows a more thorough understanding of the classifier and how it depends on the parameters of the classification task.

Definition 2 *A* model m *is a function* $m : \mathcal{X} \to \mathcal{S} \subseteq \mathbb{R}$ *from the sample space to the score space* $\mathcal{S}$ *on an unspecified scale. If it maps instances to estimates of the probability of belonging to a class, it is referred to as a* probabilistic model*. In this case* $m : \mathcal{X} \to [0, 1]$.

Definition 3 *A* threshold t *is a function* $t : \mathcal{S} \to \mathcal{Y}$ *defined by*

$$t(s) = \begin{Bmatrix} 0 & \text{if } s \leq t \\ 1 & \text{else} \end{Bmatrix}.$$

A model can be thought of as a family of classifiers parametrized by a threshold. A model and a threshold $m \circ t$ yield a classifier, in the sense that, given a predicted score $s = m(x)$, the instance x is mapped to class 1 if $s > t$, and to class 0 otherwise.

$$\begin{tikzcd}
\mathcal{X} \arrow[rr, "\delta"] \arrow[dr, "m"'] & & \mathcal{Y} \\
& \mathcal{S} \arrow[ur, "t"'] &
\end{tikzcd}$$

Every classifier defines a partition of the sample Space $\mathcal{X}$. If the classifier is binary, it divides the sample space $\mathcal{X}$ into 2 disjoint regions R_1 and R_2 that satisfy $R_1 \cap R_2 = \emptyset$. One can define R_k as

$$R_k(\delta) := \{\boldsymbol{x} \in \mathcal{X} : \delta(\boldsymbol{x}) = k\} \tag{2.1}$$

Using this notation a classifier can be restated as

$$\delta(\boldsymbol{x}) = k \text{ iff } \boldsymbol{x} \in R_k(\delta).$$

The specific form of the partition depends on the discrimination function. The simplest type of partition possible can be characterized by a single threshold on the sample space.

$$R_1(\delta_{<t}) = \{x \in \mathbb{R} : x < t\}$$

In this paper we will consider two simple types of univariate classifiers, in particular, *threshold classifiers* and *interval classifiers.*

Definition 4 (Threshold Classifier) *A classifier δ is called a* threshold classifier *if it allows the following representation*

$$\begin{aligned} \delta_{<t}(x) &:= \mathbb{1}_{(-\infty,t)}(x), \quad t \in \mathbb{R}, \\ \delta_{\geq t}(x) &:= \mathbb{1}_{[t,\infty)}(x), \quad t \in \mathbb{R}. \end{aligned} \tag{2.2}$$

The family of threshold classifiers shall be denoted by $\mathcal{F}_T := \big\{\delta_{<t}(x), t \in \mathbb{R}\big\} \cup \big\{\delta_{\geq t}(x), t \in \mathbb{R}\big\}$.

Threshold classifiers can be motivated in different ways. In Section 3 we will see an example of a parametric model which leads to a threshold classifier. The model yields the a-posteriori probability of a class membership given the observed value of X. Setting a threshold on the probability space partitions the sample space in such a way. However, more direct approaches to threshold classification will be discussed in Section 4, where the score space coincides with the sample space.

Definition 5 (Interval Classifier) *A classifier δ is called an* interval classifier *if it allows the following representation*

$$\begin{aligned} \delta_{(t_1,t_2]}(x) &:= \mathbb{1}_{(t_1,t_2]}(x), \quad t_1, t_2 \in \mathbb{R}, \\ \delta_{\mathbb{R}\setminus(t_1,t_2]}(x) &:= \mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}, \quad t_1, t_2 \in \mathbb{R}. \end{aligned} \tag{2.3}$$

The family of interval classifiers shall be denoted $\mathcal{F}_I = \{\delta_{(t_1,t_2]}(x), t_1, t_2 \in \mathbb{R}\} \cup \{\delta_{\mathbb{R}\setminus(t_1,t_2]}(x), t_1, t_2 \in \mathbb{R}\}$.

In Section 3 we will also see a parametric probabilistic model yielding the a-posteriori distribution of the classes, which will partition the sample space into an interval and its complement. A more direct nonparametric approach will be introduced in Section 5.

In the context of supervised classification, the classifier is fitted from a random sample, in which the class memberships of the instances are known. A common approach is to make a distributional assumption for the class conditional distribution and to find the optimal classifier among this family of distributions. This constitutes a parametric approach to fitting a classifier. One can also assume a specific form of classifier, which is considered a non-parametric approach. In any case one needs a formal criterion to select the optimal classifier among the chosen family. The most common approach is based on a functional called the *expected prediction error*.

Definition 6 (Expected Prediction Error)

$$EPE(\delta) := \mathbb{E}_{(X,Y)}[L(\delta(X), Y)], \tag{2.4}$$

where L denotes the loss function

$$L(\delta(X), Y) := \left\{ \begin{matrix} c_1, & \delta(X) = 0 \wedge Y = 1 \\ c_0, & \delta(X) = 1 \wedge Y = 0 \\ 0, & \delta(X) = Y \end{matrix} \right\}, \tag{2.5}$$

and $c_0 \geq 0$ and $c_1 \geq 0$ denote the misclassification costs of negative ($Y = 0$) and positive ($Y = 1$) instances, respectively.

These costs need not be monetary. Using this loss function makes two important implicit assumptions. First, the costs for an incorrect prediction of an instance depends only on its class. This is sufficient in most research situations. A more general approach would be to assign every instance its own cost. Secondly, the cost for misclassifying a positive and negative instance need not be equal. This loss function allows to weight the misclassification of positive and negative instances unequally. The third implicit assumption is that a correct classification incurs no cost.

Let us introduce the notion of an *operating condition* (OC), also referred to as *deployment condition*, see Hernández-Orallo et al. (2012) for an introduction.

Definition 7 (Operating Conditions) *The* operating conditions *consist of a triple $\theta = (c_0, c_1, \pi_1)$, which parametrizes the circumstances under which the classifier is applied. π_1 denotes the share of positives in the population ($Y \sim B_{\pi_1}$) known as the prevalence in epidemiology. Since the share of positives and negatives must add up to one, the share of negatives π_0 equals $1 - \pi_1$. c_0 and c_1 denote the misclassification costs of negative and positive instances, respectively. Thus, the space of operating conditions is $\Theta = \mathbb{R}^+ \times \mathbb{R}^+ \times [0, 1]$.*

There are different possibilities to parametrize this space. Flach (2012) parametrizes the operating space by (b, c, π_0), where $b = c_0 + c_1$ and $c = c_0/b$, referred to as the *cost proportion.*

The importance of operating conditions can be illustrated by a simple example. Consider the diagnosis of a cancer patient. Cancer is relatively rare and misdiagnosing a cancer patient has more severe consequences than misdiagnosing a healthy individual. Not taking these characteristics of the diagnosis into account when selecting a suitable model for diagnosis can lead to drawing false conclusions with possibly grave consequences.

Let us now introduce some evaluation metrics for classifiers and their empirical counterparts. Suppose $(x_i, y_i), i = 1, \dots, n$, are i.i.d. realizations of (X, Y). Let $n_1 := \sum_{i=1}^{n} y_i$ and $n_0 := n - n_1$ denote the number of positives and negatives, respectively. Without loss of generality let us assume that $y_i = 0$ for $i = 1, \dots, n_0$ and $y_i = 1$ for $i = n_0 + 1, \dots, n_0 + n_1 = n$.

Definition 8 *Given a binary classifier δ and a random variable (X, Y), the True Positive Rate* [1] *(TPR) is defined as*

$$TPR := P[\delta(X) = 1 | Y = 1].$$

In a similar fashion the False Positive Rate (FPR), the True Negative Rate[2] *(TNR) and the False Negative Rate (FNR) are defined*

$$FPR := P[\delta(X) = 1 | Y = 0],$$

$$TNR := P[\delta(X) = 0 | Y = 0],$$

$$FNR := P[\delta(X) = 0 | Y = 1],$$

respectively. Their empirical counterparts can be denoted:

$$tpr := \frac{1}{n_1} \sum_{i:y_i=1} \mathbb{1}(\delta(x_i) = 1),$$

[1] also called Sensitivity.
[2] also called Specificity.

$$fpr := \frac{1}{n_0} \sum_{i:y_i=0} \mathbb{1}(\delta(x_i) = 1),$$

$$tnr := \frac{1}{n_0} \sum_{i:y_i=0} \mathbb{1}(\delta(x_i) = 0),$$

$$fnr := \frac{1}{n_1} \sum_{i:y_i=1} \mathbb{1}(\delta(x_i) = 0).$$

Applying Bayes' theorem to Equation (2.4), we can rewrite this expression as a weighted sum of the false positive rate and the false negative rate, where the weights are a function of the operating conditions.

$$\text{EPE}(\delta) = c_0\pi_0 FPR + \ c_1\pi_1 FNR. \tag{2.6}$$

The exact derivation of (2.6) can be found in Section A.1 in the Appendix. The empirical version of this term is simply

$$\widehat{EPE}(\delta) = c_0\pi_0 fpr + \ c_1\pi_1 fnr. \tag{2.7}$$

This will be the most important evaluation metric used throughout this work. However, a great number of different metrics exist in the classification literature. The simplest loss function is characterized by an equal valuation of every misclassification ($c_0 = c_1$). This approach amounts to simply counting the number of misclassified cases and is called the *Misclassification Error.*

Definition 9 *The theoretical* Misclassification Error *(MCE) of a discrimination rule δ can be written as*

$$MCE(\delta) = \sum_{k=1}^{K} P[\delta(X) \neq k, Y = k], \tag{2.8}$$

where P is the joint distribution of (X, Y). The Empirical Classification Error *or* Misclassification Rate *based on a sample of m observations is simply the rate of incorrectly classified samples*

$$mce(\delta) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{\delta(\boldsymbol{x}_i) \neq c_i\}. \tag{2.9}$$

Definition 10 *Accuracy is defined as $ACC = \pi TPR + (1 - \pi)TNR$. This can be estimated by $\widehat{ACC} := (tp + tn)/(pos + neg)$*

Definition 11 *The ROC curve is defined as a plot between the FPR ($F^1(t)$) on the x-Axis and the TPR ($F^0(t)$) on the y-Axis as the threshold t shift through the score space. F^0 and F^1 denote the cumulative distribution function induced by the model m.*

The Area Under the ROC curve (AUC) can be defined as

$$AUC = \int_0^1 F^0(s)dF^1(s) = \int_{-\infty}^{\infty} F^0(s)f_1(s)ds,$$

where f_1 denotes the density of F^1.

While a classifier δ is depicted as one point in ROC space, a model m is depicted as a family of classifiers which are on one line between (0,0) and (1,1) (Drummond and Holte, 2006).

So far, we have considered evaluation metrics for two situations. Firstly, EPE where the operating conditions are required to be known and secondly, AUC which evaluates the classifier under the implicit assumption that all points in the parameter space Θ are equally likely. However, in many situation, the operating conditions are not know exactly but there are certain expectations, e.g. $c_1 \gg c_0$. Thus, evaluation metrics can be defined as integrals over a probability distribution over the parameter space, see e.g. Adams and Hand (1999) or Hernández-Orallo et al. (2012).

CHAPTER 3

EBC: Expected Prediction Error of the Bayes Classifier

If the class conditional distributions F^0 and F^1 are known, it is possible to derive the prediction error minimizing classifier, called the Bayes classifier, see e.g. Hastie et al. (2001). Let us define

$$EBC := EPE(\delta_{Bayes}), \tag{3.1}$$

where

$$\delta_{Bayes} := \min_{\delta} EPE(\delta) = \min_{\delta} \mathbf{E}_{(X,Y)}[\mathrm{L}(\delta(X), Y)], \tag{3.2}$$

and L denotes the loss functional defined in Equation (2.5). The Bayes classifier is more of a theoretical concept since the distributions of the class conditionals are rarely known in practice. However, its defining characteristic can be exploited to define a benchmark prediction error for a random variable with known distribution. Let us derive an expression for this classifier by minimizing the expected prediction error (2.4). If we condition on Y we can write EPE as

$$EPE(\delta) = \mathbf{E}_X \left[L(\delta(X), Y = 0)P(Y = 0|X) + L(\delta(X), Y = 1)P(Y = 1|X) \right].$$

This expression, can be minimized point-wise, yielding

$$\delta_{Bayes}(x) = \arg \min_{y \in \{0,1\}} \underbrace{L(\delta(X), Y = y)}_{c_y} P[Y = y|X = x] \qquad \forall x \in \mathcal{X} \subseteq \mathbb{R}^p. \tag{3.3}$$

By applying Bayes' theorem for discrete Y and continuous X

$$P[Y = y|X] = \frac{f_{X|Y=y}\pi_y}{f_{X|Y=0}\pi_0 + f_{X|Y=1}\pi_1}$$

to express the probability conditional to X as a function of the class conditional probabilities, then $\forall x \in \mathcal{X} \subseteq \mathbb{R}^p : P[Y=0|X=x] > 0$ we obtain the following classifier

$$\delta_{Bayes}(x) = \begin{Bmatrix} 1 & \text{if } \frac{f_{X|Y=1}(x)}{f_{X|Y=0}(x)} \geq \frac{\pi_0}{(1-\pi_0)} \cdot \frac{c_0}{c_1} \\ 0 & \text{else} \end{Bmatrix}. \tag{3.4}$$

From Equation (3.4) we can see that the Bayes classifier can be conceived as a model, as defined in Definition (2), where $m(x) = \frac{f_{X|Y=1}(x)}{f_{X|Y=0}(x)}$ and the threshold is a function of the operating conditions $t(c_0, c_1, \pi_1) = \frac{\pi_0}{(1-\pi_0)} \frac{c_0}{c_1}$. This helps us understand how the OCs influence the partition of the classifier.

If the class conditional distributions are Gaussian $(X|Y=1) \sim N(\mu_1, \sigma_1^2)$ and $(X|Y=0) \sim N(\mu_0, \sigma_0^2)$ with the corresponding densities $f(x;\mu_0,\sigma_0)$ and $f(x;\mu_1,\sigma_1)$, the classifier is equivalent to a univariate Quadratic Discriminant Analysis (for $\sigma_0 \neq \sigma_1$), or a univariate *Linear Discriminant Analysis* (for $\sigma_0 = \sigma_1$), respectively.

The following theorem derives the discriminant function of LDA and QDA for the univariate case and is, thus, simply a special case of results from e.g. Example 4.2.2 in Bickel and Doksum (2015). It should motivate the use of threshold and interval classifiers.

Theorem 1 *Consider the class conditional distributions $(X|Y=1) \sim N(\mu_1, \sigma_1^2)$ and $(X|Y=0) \sim N(\mu_0, \sigma_0^2)$. If $\sigma_0^2 = \sigma_1^2$, the Bayes classifier is a threshold classifier*

$$\delta_{Bayes}(x) = \begin{cases} \delta_{<t}(x) & \text{if } \mu_0 \leq \mu_1 \\ \delta_{\geq t}(x) & \text{if } \mu_0 > \mu_1 \end{cases},$$

where $t \in \mathbb{R}$ is the solution to a linear equation.

If, however, $\sigma_0^2 \neq \sigma_1^2$, then the Bayes classifier is a member of the family of interval classifiers

$$\delta_{Bayes}(x) = \begin{cases} \delta_{(t_1,t_2]}(x) & \text{if } \sigma_1 \leq \sigma_0 \\ \delta_{\mathbb{R}\setminus(t_1,t_2]}(x) & \text{if } \sigma_1 > \sigma_0 \end{cases},$$

where $t_1 \in \mathbb{R}$ and $t_2 \in \mathbb{R}$ are the solutions of a quadratic equation.

Proof 1 *Let us insert the densities of the Gaussian CCDs in* (3.4) *and logarithmise both sides, yielding*

$$\ln\left(\frac{f(x;\mu_1,\sigma_1)}{f(x;\mu_0,\sigma_0)}\right) = \frac{(x-\mu_0)^2}{2\sigma_0^2} - \frac{(x-\mu_1)^2}{2\sigma_1^2} + \ln\left(\frac{\sigma_0}{\sigma_1}\right) \geq \ln\left(\frac{\pi_0}{(1-\pi_0)} \cdot \frac{c_0}{c_1}\right). \tag{3.5}$$

If $\sigma_0 = \sigma_1$ then x^2 cancels out and (3.5) *is a linear function in x and its root equals*

$$t = \ln\left(\frac{c_0}{c_1}\frac{\pi_0}{\pi_1}\right)\frac{\sigma^2}{(\mu_1 - \mu_0)} - \frac{(\mu_1 + \mu_2)}{2}. \tag{3.6}$$

Thus, the resulting Bayes classifier is given by

$$\delta_{Bayes}(x) = \begin{cases} \mathbb{1}_{(t,\infty)}(x) \text{ if } \mu_0 \leq \mu_1 \\ \mathbb{1}_{(-\infty,t]}(x) \text{ if } \mu_0 > \mu_1 \end{cases}.$$

In the general case ($\sigma_0 \neq \sigma_1$), Equation (3.5) is quadratic in x. If we define $a := \frac{1}{2\sigma_0^2} - \frac{1}{2\sigma_1^2}$, $b := \frac{\mu_1}{\sigma_1^2} - \frac{\mu_0}{\sigma_0^2}$, and $c := \frac{\mu_0^2}{2\sigma_0^2} - \frac{\mu_1^2}{2\sigma_1^2} + \ln(\frac{\sigma_0}{\sigma_1}) - \ln(\frac{\pi_0}{\pi_1}\frac{c_0}{c_1})$, the equation can be rewritten as $ax^2 + bx + c = 0$. If $b^2 - 4ac < 0$ there is no real valued solution and the Bayes Classifier becomes degenerate in the sense that it maps to one class only, irrespective of the value of X. If $b^2 - 4ac > 0$, the number of solutions is two, t_1 and t_2. In this case the Bayes Classifier can be denoted

$$\delta_{Bayes}(x) = \begin{cases} \mathbb{1}_{(t_1,t_2]}(x) \text{ if } \sigma_0 < \sigma_1 \\ \mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(x) \text{ if } \sigma_0 \geq \sigma_1 \end{cases}. \tag{3.7}$$

□

An illustration of the Bayes classifier for the variable Z_3 of the example in Section 1 can be found in Figure 3.1.

Let us derive the corresponding prediction error to the Bayes classifier. If $\sigma_0 = \sigma_1$, then

$$EBC = \begin{Bmatrix} c_1\pi_1\Phi^1(t) + c_0\pi_0\left(1 - \Phi^0(t)\right) & \text{iff } \mu_0 \leq \mu_1 \\ c_1\pi_1\left(1 - \Phi^1(t)\right) + c_0\pi_0\Phi^0(t) & \text{else} \end{Bmatrix}, \tag{3.8}$$

where Φ^0 and Φ^1 are the Gaussian cumulative distribution functions of X^0 and X^1 respectively. Since the cumulative distribution function of the Gaussian distribution does not have a closed form expression and can only be written as an integral of the Gaussian density, no closed form expression of EBC exists.

If $\sigma_0 \neq \sigma_1$, then

$$EBC = \begin{Bmatrix} c_1\pi_1\left[\Phi^1(t_2) - \Phi^1(t_1)\right] + c_0\pi_0\left[1 - (\Phi^0(t_2) - \Phi^0(t_1))\right] & \text{if } \sigma_0 \geq \sigma_1 \\ c_1\pi_1\left[1 - (\Phi^1(t_2) - \Phi^1(t_1))\right] + c_0\pi_0\left[\Phi^0(t_2) - \Phi^0(t_1)\right] & \text{else} \end{Bmatrix}. \tag{3.9}$$

Again, no closed form expression can be given.

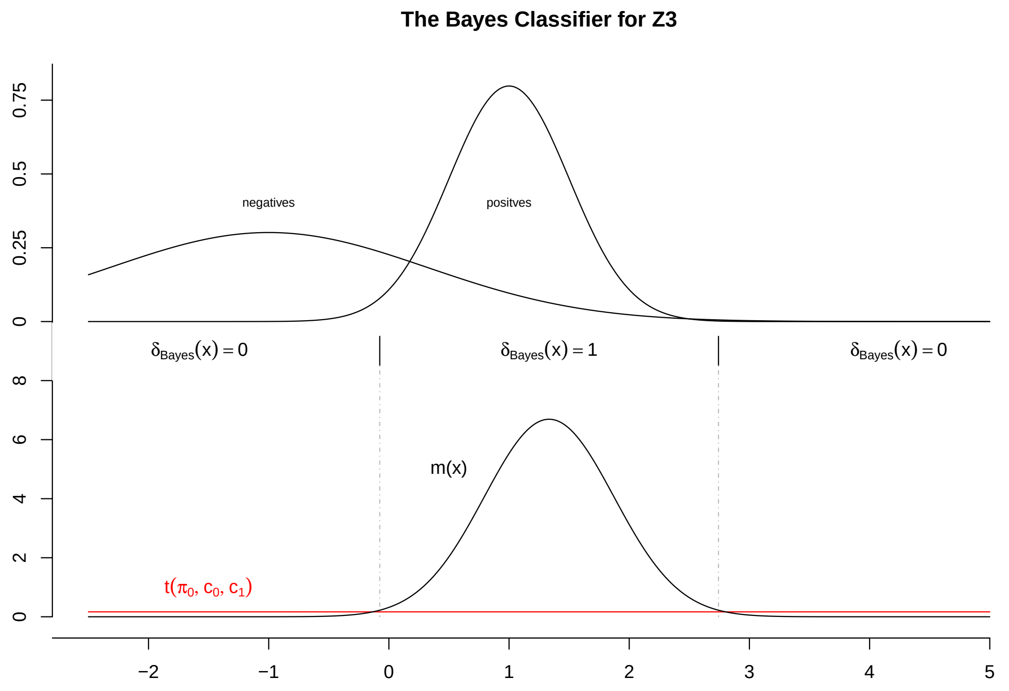


Figure 3.1: Illustration of the Bayes classifier of $(X|Y = 0) \sim N(-0.5, \sqrt{7/8})$ and $(X|Y = 1) \sim N(0.5, \sqrt{1/8})$. This corresponds to Z_3 of the example in Figure 1.1 in Section 1. The upper image depicts the distribution of the positive and negative class and the interval, in which the Bayes classifier maps instances to the positive class. The lower image depicts the model $m(x) = \frac{f_{X|Y=1}(x)}{f_{X|Y=0}(x)}$ and the threshold $t(c_0, c_1, \pi_1) = \frac{\pi_0}{(1-\pi_0)} \frac{c_0}{c_1}$ as defined in (2) and (3). This image illustrates how the positive domain $\{x \in \mathbb{R} : m(x) \geq t(c_0, c_1, \pi_0)\}$ depends on the operating conditions of the classification task.

3.1 Sample Estimate of EBC

Consider the situation where the class conditional distributions $P(X|Y=1)$ and $P(X|Y=0)$ are assumed to be Gaussian, their parameters (μ_1, σ_1^2) and (μ_0, σ_0^2) are, however, unknown. Under this set of assumptions, the expected prediction error can be obtained by the following steps.

- Estimate the parameters of the Gaussian distributions, e.g. by using the maximum likelihood estimators or a robust equivalent yielding $(\hat{\mu}_0, \hat{\sigma}_0^2)$ and $(\hat{\mu}_1, \hat{\sigma}_1^2)$. If equal variances are assumed, the entire sample can be used for its estimate.

- Derive the corresponding Bayes classifier. If we assume equal variances of the CCDs this will be

$$\delta_{Bayes} = \begin{cases} \delta_{<\hat{t}} \text{ if } \hat{\mu}_1 < \hat{\mu}_0 \\ \delta_{\geq \hat{t}} \text{ if } \hat{\mu}_1 \geq \hat{\mu}_0 \end{cases},$$

  where $\hat{t} = t(\hat{\mu}_0, \hat{\mu}_1, \hat{\sigma})$ defined in (3.6). If this assumption is relaxed

$$\delta_{Bayes}(x) = \begin{cases} \delta_{(\hat{t}_1, \hat{t}_2]}(x) & \text{if } \hat{\sigma}_1 \leq \hat{\sigma}_0 \\ \delta_{\mathbb{R}\setminus(\hat{t}_1, \hat{t}_2]}(x) & \text{if } \hat{\sigma}_1 > \hat{\sigma}_0 \end{cases},$$

  where $\hat{t}_1 \in \mathbb{R}$ and $\hat{t}_2 \in \mathbb{R}$ are the solutions of the quadratic equation defined in (3.7). Bare in mind, that if the decision to assume equal variances or not is based on the data, the estimators are not unbiased any more.

- Calculate the prediction error defined in (2.6). Again, we need to distinguish between two cases. For $\sigma_0 = \sigma_1$ we will use Equation (3.8) and plug in the estimates, yielding

$$\widehat{EBC} = \begin{cases} c_1\pi_1\Phi(\hat{t}; \hat{\mu}_1, \hat{\sigma}^2) + c_0\pi_0\left(1 - \Phi(\hat{t}; \hat{\mu}_0, \hat{\sigma}^2)\right) & \text{iff } \hat{\mu}_0 \leq \hat{\mu}_1 \\ c_1\pi_1\left(1 - \Phi(\hat{t}; \hat{\mu}_1, \hat{\sigma}^2)\right) + c_0\pi_0\Phi(\hat{t}; \hat{\mu}_0, \hat{\sigma}^2) & \text{else} \end{cases}.$$

  For $\sigma_0 \neq \sigma_1$ we will use Equation (3.9) and plug in all estimates, yielding

$$\widehat{EBC} = \begin{cases} c_1\pi_1\left[\Phi(\hat{t}_2; \hat{\mu}_1, \hat{\sigma}_1) - \Phi(\hat{t}_1; \hat{\mu}_1, \hat{\sigma}_1)\right] + c_0\pi_0\left[1 - (\Phi(\hat{t}_2; \hat{\mu}_0, \hat{\sigma}_0) - \Phi(\hat{t}_1; \hat{\mu}_0, \hat{\sigma}_0))\right] & \text{if } \hat{\sigma}_0 \geq \hat{\sigma}_1 \\ c_1\pi_1\left[1 - (\Phi(\hat{t}_2; \hat{\mu}_1, \hat{\sigma}_1) - \Phi(\hat{t}_1; \hat{\mu}_1, \hat{\sigma}_1))\right] + c_0\pi_0\left[\Phi(\hat{t}_2; \hat{\mu}_0, \hat{\sigma}_0) - \Phi(\hat{t}_1; \hat{\mu}_0, \hat{\sigma}_0)\right] & \text{else} \end{cases}.$$

3.2 Derivation of the Null Distribution

The analytic derivation of the (asymptotic) null distribution of $\widehat{EBC}$ is beyond the scope of this thesis and, thus, we have to resort to a resampling approach. A well established method for this purpose is by permuting the class labels, see e.g. Edgington

(2011). Under the assumption of exchangeability of the observations, which is weaker than independence, the permuted values of the classes follow the same distribution. The empirical cumulative distribution function of the resulting values will converge to the true null distribution if H_0 is valid. This approach, however, does not yield a good estimate of the extreme tail of the null distribution, where for the purpose of screening the most interesting values lie. Even if we draw 10^5 random permutations, the expected number of observations below the 0.0001 quantile will be too low for a good estimate of the density. Knijnenburg et al. (2009) have attempted to approximate the tails by means of a generalized Pareto distribution. This, however, did not yield satisfying results. We, thus, used a monotonic interpolation, introduced by Hyman (1983), between the point (0,0) and the smallest values obtained by permutation to obtain an estimate of the tail. Even if this approach does not provide a good estimate of the tail, it yields a monotonically and continuously decreasing function of x and, thus, variables with a lower statistic will yield a lower p-value and the order of the variables is maintained.

CHAPTER 4

ETC: Expected Prediction Error of the Threshold Classifier

In this section a non-parametric equivalent of EBC for equal variances ($\sigma_0 = \sigma_1$) will be introduced and studied. In Theorem 1 we have learned that for Gaussian class conditionals with equal variances a threshold classifier is optimal. In this section we will opt for a more direct and nonparametric approach to deriving a classifier. Instead of making a distributional assumption, we will limit our search to the family of threshold classifiers, defined in (4), and find the best member using the sample error.

Let us define ETC, the expected prediction error of the optimal threshold classifer

$$ETC := \min_{\delta \in \mathcal{F}_T} EPE(\delta). \tag{4.1}$$

This statistic estimates the prediction error resulting from separating the two classes by simply setting a threshold on the sample space of X. It ranges from 0, when perfect separation is possible, to $\min(c_0\pi_0, c_1\pi_1)$, when X exhibits no discriminatory power with respect to the classes of Y.

4.1 Sample Estimate of ETC

Suppose $(x_i, y_i), i = 1, \ldots, n$, are i.i.d. realizations of (X, Y) and let $n_1 := \sum_{i=1}^n y_i$ and $n_0 := n - n_1$ denote the number of positives and negatives, respectively. Without loss of generality let us assume that $y_i = 0$ for $i = 1, \ldots, n_0$ and $y_i = 1$ for $i = n_0{+}1, \ldots, n_0{+}n_1 = n$. To derive a sample estimate of ETC we simply need to substitute the false positive and false negative rates in (2.6) by their empirical counterparts (2.7). Due to the specific form of a threshold classifier its false negative rate $P[\delta_{<t}(X) = 1|Y = 0]$ is simply $F^0(t)$, the cumulative CCD function evaluated at t. By the same argument

$P[\delta_{<t}(X)=0|Y=1]=1-F^1(t)$ and, thus, $EPE(\delta_{<t})$ is simply a weighted sum of $F^0(t)$ and $F^1(t)$. Substituting these expressions by their empirical counterpart, yields

$$\begin{aligned}\widehat{EPE}(\delta_{<t}) &= \underbrace{c_0\pi_0\frac{1}{n_0}\sum_{i:y_i=0}\mathbb{1}_{(-\infty,t)}(x_i)}_{\text{false positive rate}}+\underbrace{c_1\pi_1\frac{1}{n_1}\sum_{i:y_i=1}\mathbb{1}_{[t,\infty)}(x_i)}_{\text{false negative rate}}\\ \widehat{EPE}(\delta_{\geq t}) &= \underbrace{c_0\pi_0\frac{1}{n_0}\sum_{i:y_i=0}\mathbb{1}_{[t,\infty)}(x_i)}_{\text{false positive rate}}+\underbrace{c_1\pi_1\frac{1}{n_1}\sum_{i:y_i=1}\mathbb{1}_{(-\infty,t)}(x_i)}_{\text{false negative rate}}.\end{aligned} \tag{4.2}$$

Note that $\widehat{EPE}(\delta_{<t})$ is constant for all thresholds $t\in[x_{(j)},x_{(j+1)})$, where $x_{(j)}$ denotes the j-th order statistic of the sample. Thus, instead of having to search $\mathbb{R}$ it suffices to find the minimum of $n+1$ values. Exploiting this fact leads us to the empirical counterpart of (4.1),

$$\widehat{ETC}=\min\left\{\min_{j=1,\ldots,n}\widehat{EPE}(\delta_{<x_j}),\min_{j=1,\ldots,n}\widehat{EPE}(\delta_{\geq x_j})\right\}. \tag{4.3}$$

The method can more easily be understood when it is described in the following algorithmic manner. This algorithm builds on the fact that the sample estimate of the prediction error of a variable is equal for all threshold classifiers $\delta_{\lesseqgtr t}: t\in[x_{(i)},x_{(i+1)})$, where $x_{(i)}$ indicates the i-th order statistic of the sample. Thus, instead of having to search $\mathbb{R}$ it suffices to find the minimum of the estimated prediction errors of $2n$ threshold classifiers. An illustration of this procedure can be found in Figure 4.1.

ETC

Consider the data $(x_i,y_i)_{i=1,\ldots,n}$.

- Sort $(x_i,y_i), i=1,\ldots,n$ in increasing order, such that $x_{(1)}\leq x_{(2)}\leq\ldots\leq x_{(n)}$.
- Calculate $\widehat{EPE}(\delta_{<x_{(i)}})$ and $\widehat{EPE}(\delta_{\geq x_{(i)}})$ for $i=1,\ldots,n$ and
- find the minimal value

$$\widehat{ETC}=\min\left\{\min_{i=1,\ldots,n}\widehat{EPE}(\delta_{<x_{(i)}}),\min_{i=1,\ldots,n}\widehat{EPE}(\delta_{\geq x_{(i)}})\right\}.$$

4.2 Properties of $\widehat{ETC}$

This chapter will establish some properties of the sample estimate of ETC. Firstly, $\widehat{ETC}$ is not an unbiased estimator of ETC, even though, $\widehat{EPE}(\delta_{<t})$ is an unbiased estimator

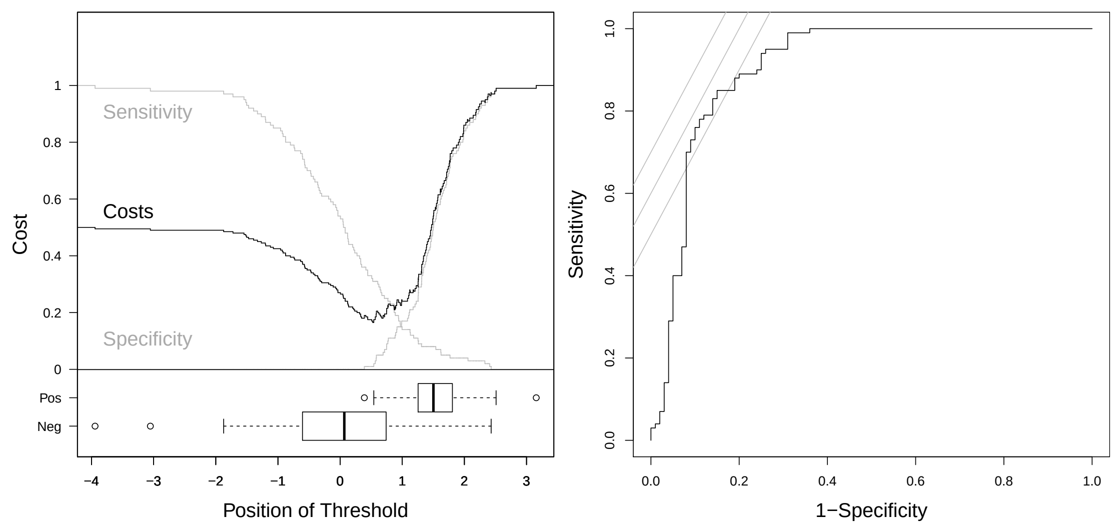


Figure 4.1: Illustration of the ETC algorithm. The graph on the left hand side shows the relationship between the threshold t of the classifier $\delta_{<t}$ and the expected prediction error. As the threshold slides through the sample space the number of false positives and false negatives change. This is depicted in the sensitivity and specificity curves. The costs, which are a weighted average of the sensitivity and the specificity, have a unique minimum. The graph on the right depicts the curve of all pairs of sensitivity and 1 - specificity as the threshold slides through the sample space. These pairs shape the ROC curve of the classifier. The gray lines are called the isocost lines. They indicate all points in the sensitivity, 1 - specificity space that generate the same costs. The "higher" the isocost line, the lower the costs. The optimal point is where the ROC curve touches the highest isocost line.

of $EPE(\delta_{<t})$,

$$\begin{aligned}\mathbf{E}\left[\widehat{EPE}(\delta_{<t})\right] &= c_0\pi_0\tfrac{1}{n_0}\textstyle\sum_{i:y_i=0}\mathbf{E}\mathbb{1}_{(-\infty,t)}(x_i) + \; c_1\pi_1\tfrac{1}{n_1}\textstyle\sum_{i:y_i=1}\mathbf{E}\mathbb{1}_{[t,\infty)}(x_i) = \\ &= c_0\pi_0F^0(t) + c_1\pi_1\left(1 - F^1(t)\right),\end{aligned}$$

since $\mathbb{1}_{(-\infty,t)}(X^0) \sim \mathcal{B}_{1,F^0(t)}$ and $\mathbb{1}_{[t,\infty)}(X^0) \sim \mathcal{B}_{1,1-F^0(t)}$. Let us assume without loss of generality that δ_{t^*} yields the minimal EPE among all threshold classifiers. Trying to estimate t^* by the sample point x_j that yields the minimal $\widehat{EPE}(\delta_{<x_j})$ over all instances and both directions introduces a bias. Since there will always exist an index $i \in 1,\dots,n$ such that $\widehat{EPE}(\delta_{<x_i}) = \widehat{EPE}(\delta_{<t^*})$, the following inequality holds

$$\mathbf{E}\left[\min_{j=1,\dots,n}\widehat{EPE}(\delta_{<x_j})\right] \leq \mathbf{E}\left[\widehat{EPE}(\delta_{<t^*})\right].$$

Thus, the true ETC is an upper bound of our test statistic. We can verify that there

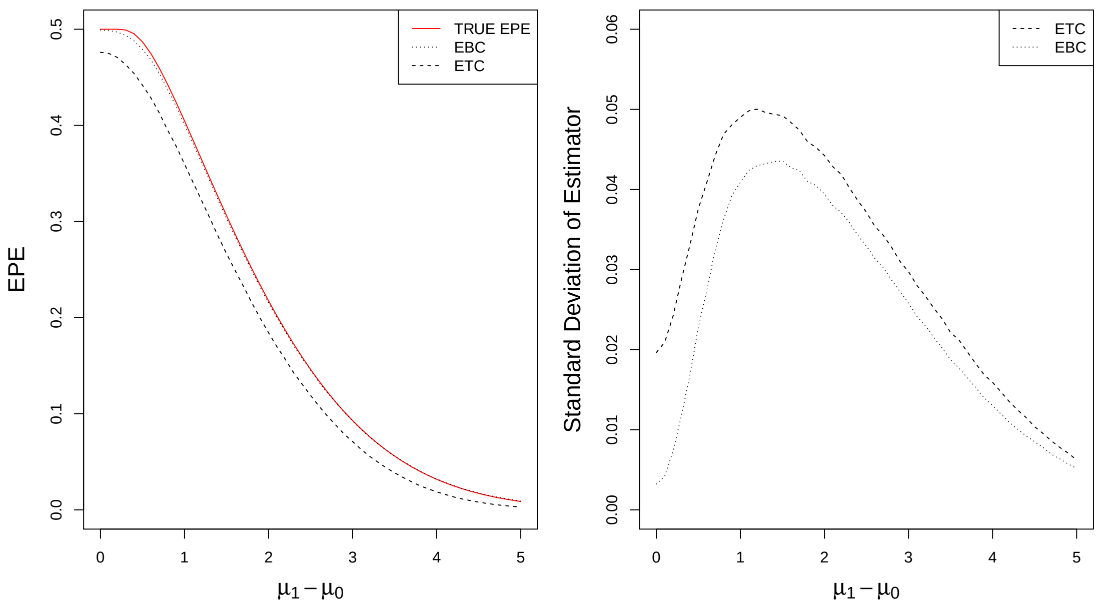


Figure 4.2: The left Figure illustrates the true EPE as a function of the difference of the means of two Gaussian CCDs and the expected value of its estimations by EBC and ETC. The variance of the two distributions is equal, thus, EBC represents the prediction error of the Bayes classifier and is, thus, minimal among all possible classifiers. One can clearly see that ETC is not an unbiased estimator. The right figure illustrates the variances of the two estimators $\widehat{EBC}$ and $\widehat{ETC}$.

exists a negative bias by means of simulation. The results of one such study are illustrated in Figure 4.2.

$$\text{Bias}(\widehat{ETC}) = \mathbf{E}_{(X,Y)}\left[\widehat{ETC}\right] - EPE_{(X,Y)}(\delta_{Bayes}) < 0$$

In order to evaluate the bias of $\widehat{ETC}$ a simulation study was conducted. For this purpose random numbers from two Gaussian class conditionals with varying differences in the means were drawn and the true expected prediction errors of the corresponding Bayes classifiers were calculated from the known parameters. This was compared to the mean estimates of EBC and ETC. The results can be found in Figure 4.2.

The following theorem will establish the consistency of $\widehat{ETC}$.

Theorem 2 *Assume continuous class conditional distributions F^0 and F^1 and a unique minimizer of ETC, then $\widehat{ETC}$ is a consistent estimator of ETC,*

$$plim_{n\to\infty}\widehat{ETC}_n = ETC.$$

Proof 2 *Consider an independent and identically distributed sample* $(X_i, Y_i)_{i=1,\dots,n}$*, where* $X_i|Y_i = 0 \sim F^0$ *and* $X_i|Y_i = 1 \sim F^1$*. To establish the consistency, we have to show that*

$$\forall \epsilon > 0 : \lim_{n\to\infty} P\left[\left|\widehat{ETC}_n - ETC\right| > \epsilon\right] = 0. \tag{4.4}$$

Without loss of generality assume that $\delta_{<t^*}$ *minimizes the prediction error among all threshold classifiers*

$$\delta_{<t^*} = \operatorname*{argmin}_{\delta\in\mathcal{F}_t} EPE(\delta)$$

and, thus,

$$ETC = \min_{\delta\in\mathcal{F}_t} EPE(\delta) = EPE(\delta_{<t^*}) = c_0\pi_0F^0(t^*) + c_1\pi_1(1 - F^1(t^*)).$$

Inserting this expression is (5.4) *yields*

$$\forall \epsilon > 0 : \lim_{n\to\infty} P\left[\left|\widehat{ETC}_n - \left(c_0\pi_0F^0(t^*) + c_1\pi_1(1 - F^1(t^*))\right)\right| > \epsilon\right] = 0.$$

Let us start with the definition of $\widehat{ETC}$*, perceived as a random variable*

$$\widehat{ETC}_n = \min\left\{\min_{j=1,\dots,n} \widehat{EPE}(\delta_{<X_j}), \min_{j=1,\dots,n} \widehat{EPE}(\delta_{\geq X_j})\right\}.$$

The minimum is a continuous function and, thus, by virtue of the continuous mapping theorem all we need to show is the consistency of its arguments. Due to the assumptions made about $\delta_{<t^*}$ *we will show the convergence of the first argument. As* $n \to \infty$*, both* $n_0, n_1 \to \infty$ *at rates* $n_1 \approx \pi_1 n$ *and* $n_0 \approx \pi_0 n$*. Further, let us denote* $\hat{F}^0_{n_0}(x) := \frac{1}{n_0}\sum_{i:Y_i=0} \mathbb{1}_{(-\infty,x)}(X_i)$ *and* $\hat{F}^1_{n_1}(x) := \frac{1}{n_1}\sum_{i:Y_i=1} \mathbb{1}_{(-\infty,x)}(X_i)$*. The problem then translates to showing that* $\forall\epsilon > 0$:

$$\lim_{n\to\infty} P\left[\left|\min_{j=1,\dots,n} c_0\pi_0\hat{F}^0_{n_0}(X_j) + c_1\pi_1\left(1 - \hat{F}^1_{n_1}(X_j)\right) - \left(c_0\pi_0F^0(t^*) + c_1\pi_1(1 - F^1(t^*))\right)\right| > \epsilon\right] = 0$$

By virtue of the Glivenko-Cantelli theorem, the empirical distribution functions converge almost surely uniformly to the true distribution functions $\sup_{x\in\mathbb{R}} |\hat{F}^0_{n_0}(x) - F^0(x)| \xrightarrow{a.s.} 0$ *and* $\sup_{x\in\mathbb{R}} |\hat{F}^1_{n_1}(x) - F^1(x)| \xrightarrow{a.s.} 0$ *and, thus,*

$$\sup_{x\in\mathbb{R}}\left|\underbrace{c_0\pi_0\hat{F}^0_{n_0}(x) + c_1\pi_1\left(1 - \hat{F}^1_{n_1}(x)\right)}_{Q_n:=} - \underbrace{c_0\pi_0F^0(x) + \left(1 - F^1(x)\right)}_{Q:=}\right| \xrightarrow{a.s.} 0,$$

will also converge uniformly almost surely.

Note that since F^0 *and* F^1 *are continuous, so is* Q*, which is a nonrandom function with a unique minimum at* t^**. Further, let us denote the minimizer of* Q_n *as*

$$\hat{t}_n := \arg\min_{t\in\mathbb{R}} Q_n(t) = \min_{j=1,\ldots,n} Q_n(X_j).$$

The second equation is valid since Q_n is a step function.

We will now argue that under these circumstances the minimizer of Q_n will converge in probability to the minimizer of Q.

For every $\epsilon > 0$ we have

$$c(\epsilon) := \inf_{t\in\Theta:||t-t^*||\geq\epsilon} Q(t) > Q(t^*)$$

since $\{t \in \Theta \subseteq \mathbb{R} : ||t - t^*|| \geq \epsilon\}$ is compact and $Q(t)$ is continuous, and since t^* is the unique minimizer of Q. Choose $0 < \delta < 1/2(c(\epsilon) - Q(t^*))$, i.e., δ is such that

$$c(\epsilon) - \delta > Q(t^*) + \delta.$$

Note that $\delta = \delta(\epsilon)$. On the event

$$\left\{\sup_{t\in\Theta} |Q_n(t) - Q(t)| < \delta\right\}$$

we then have

$$\inf_{t\in\mathbb{R}:||t-t^*||\geq\epsilon} Q_n(t) \geq \inf_{t\in\Theta:||t-t^*||\geq\epsilon} Q(t) - \delta = c(\epsilon) - \delta > Q(t^*) + \delta \geq Q_n(t^*) \geq Q_n(\hat{t}_n)$$

That is, on the above event we obtain that

$$||\hat{t}_n - t^*|| < \epsilon.$$

In other words,

$$\left\{||\hat{t}_n - t^*|| < \epsilon\right\} \supseteq \left\{\sup_{t\in\Theta} |Q_n(t) - Q(t)| < \delta\right\}.$$

But

$$P\left(\sup_{t\in\Theta} |Q_n(t) - Q(t)| < \delta\right) \to 1,$$

and hence

$$P\left(||\hat{t}_n - t^*|| < \epsilon\right) \to 1$$

which establishes the required consistency.

□

An obvious and serious limitation of ETC as a ranking statistic is the low number of values it can take. This might be a limiting factor when the sample size is small and the number of variables is huge. The number depends mainly on n_1 and n_0, but also on c_1, c_0, π_0, and π_1. The fewest values are obtained when $c_0 = c_1$ and $\pi_0 = \pi_1$. In this case the number of false positives and false negatives cannot exceed $n_0/2$, and $n_1/2$, respectively, and the number of possible values for ETC is $\max\{n_0/2, n_1/2\}$. Even when e.g. $c_1 > n_0 c_0$, when one positive outweighs all negatives the number of possible values $n_0 n_1$ is limited when $p \gg n$.

Furthermore, unlike methods which are not based on the ranks of X, $\widehat{ETC}$ cannot differentiate between variables, which exhibit a perfect discrimination, since it will assign all those variables the value 0 and the same p-value.

4.3 Derivation of the Null Distribution

We might pose the question of the significance of a certain value of $\widehat{ETC}$. In other words, what is the probability of the observed or a more extreme outcome under the null hypothesis that X exhibits no information about the class membership. The null hypothesis and the alternative hypothesis can be stated $H_0 : F^0 = F^1$ and $H_1 : F^0 \neq F^1$, respectively. For the purpose of filtering, providing a p-value gives additional information, since it allows not only to rank the variables but also to decide how many variables should be selected.

An example of a null distribution for $n_0 = 9$, $n_1 = 9$, $c_0 = 1$, $c_1 = 2$, $\pi_1 = 0.5$ can be found in Figure 4.3. We will use this example in the following section to explain key features of the algorithm.

4.3.1 Independence of the CCDs under H0

The first result that we would like to establish is that the distribution of $\widehat{ETC}$ under the null hypothesis is independent of the CCDs. This important property allows us to calculate the ND once only for any number of CCDs since it only depends on the operating conditions and the sample size. Let us formulate this in the following theorem.

Theorem 3 *Consider the i.i.d. samples of the class conditional random variables $X_i^0 \sim F^0, i = 1, \dots, n_0$ and $X_i^1 \sim F^1, i = n_0+1, \dots, n$. Under the null hypothesis $H_0 : F^0(x) = F^1(x) = F(x), \forall x \in \mathcal{X}$ the sampling distribution of $\widehat{ETC}$ is independent of F.*

Proof 3 *Let us introduce r which maps a sample $(x_i, y_i)_{i=1,\dots,n}$ to the permutation of class labels ordered by increasing value of X*

$$r : (\mathcal{X} \times \{0,1\})^n \mapsto \mathcal{P}_{n_1,n_0} : r((x_i, y_i)_{i=1,\dots,n}) = (y_{(1)}, y_{(2)}, \dots, y_{(n)}), \tag{4.5}$$

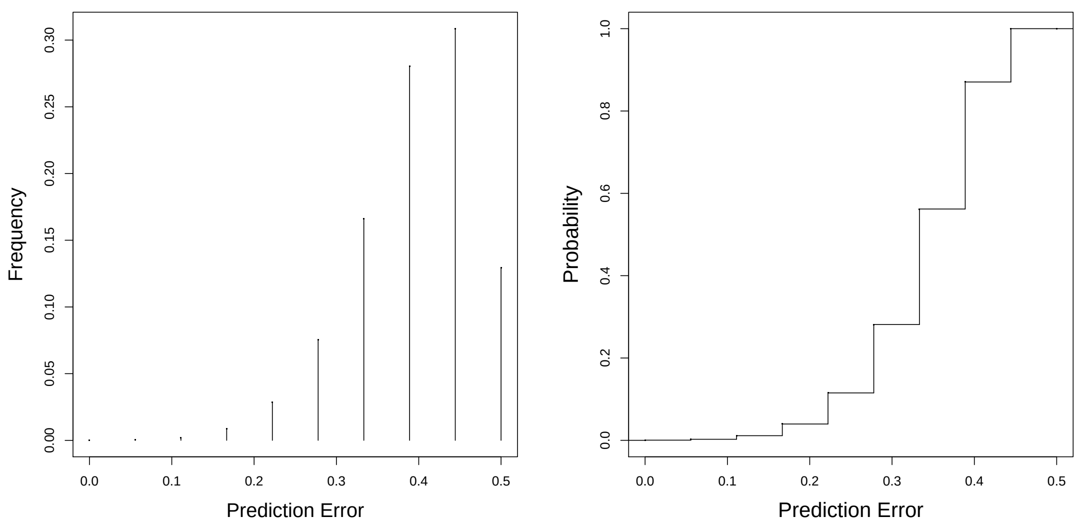


Figure 4.3: The null distribution (on the left) and the cumulative null distribution (on the right) of $\widehat{ETC}$ for the following operating conditions and sample sizes $n_0 = 9$, $n_1 = 9$, $c_0 = 1$, $c_1 = 2$, $\pi_1 = 0.5$.

where $y_{(i)}$ denotes the class label of the i-th order statistic. $\mathcal{P}_{n_1,n_0}$ denotes the space of all possible permutations of n_1 positive and n_0 negative class labels

$$\mathcal{P}_{n_1,n_0} := \left\{ (y_1, \ldots, y_n) : y_i \in \{0,1\} \wedge \sum_{i=1}^{n} y_i = n_1 \right\}.$$

The importance of r lies in the fact that although it reduces the information of the sample it still contains all the information needed to calculate $\widehat{ETC}$. This can be shown by the following factorization $\widehat{ETC} \equiv \widehat{ETC}_{\mathcal{P}} \circ r$.

$$\begin{tikzcd}
(\mathcal{X}, \{0,1\})^n \arrow[rr, "\widehat{ETC}"] \arrow[dr, "r"'] & & \mathbb{R}^+ \\
& \mathcal{P}_{n_1,n_0} \arrow[ur, "\widehat{ETC}_{\mathcal{P}}"'] &
\end{tikzcd}$$

To prove the validity of this factorization we can rewrite the first argument of (4.3)

$$\begin{aligned}
\min_{i=1,\ldots,n} \widehat{EPE}(\delta_{<x_{(i)}}) &= \min_{i=1,\ldots,n} c_0\pi_0 \frac{1}{n_0} \sum_{j=1}^{n_0} \mathbb{1}_{(-\infty,x_{(i)})}(x_j) + c_1\pi_1 \frac{1}{n_1} \sum_{j=n_0+1}^{n} \mathbb{1}_{[x_{(i)},\infty)}(x_j) \\
&= \min_{i=1,\ldots,n} c_0\pi_0 \frac{1}{n_0} \sum_{j=1}^{i} y_{(j)} + c_1\pi_1 \frac{1}{n_1} \sum_{j=i+1}^{n} (1 - y_{(j)}), \qquad (4.6)
\end{aligned}$$

as a function of $r((x_i, y_i)_{i=1,\dots,n})$*. The same holds true for the second argument in* (4.3) *and, thus, we have the required factorization of* $\widehat{ETC}$*.*

Under the null hypothesis we are effectively drawing independently n *times from the same distribution. The i.i.d. condition implies exchangeability and, thus, every order of positives and negatives is equally likely. Irrespective of the distribution of* (X, Y) *on the sample space, the induced probability distribution of* $r((X_i, Y_i)_{i=1,\dots,n})$ *on* $\mathcal{P}_{n_1,n_0}$ *is always uniform under* H_0 *and, thus, independent of* F_0 *and* F_1*.*

$$\mathbb{P}[r((X_i, Y_i)_{i=1,\dots,n}) = p] = \frac{1}{\binom{n}{n_0}} \quad \forall p \in \mathcal{P}_{n_1,n_0}$$

As a consequence, the probability distribution on $\mathbb{R}^+$ *induced by* $\widehat{ETC}_{\mathcal{P}}$ *must also be independent of* F_0 *and* F_1 *and is identical to the distribution induced by* $\widehat{ETC}$*.*

□

Let us comment on some aspects of this theorem. First, the result of this theorem is not surprising. It is a known fact that ranking statistics also exhibit the feature of independence of the distribution of the random variable under the assumption of exchangeability. The statistic r is related to the ranks in the sense that it represents the vector of class labels sorted according to the ranks of x_i.

Secondly, the argument that the vector $r(x_i, y_i)$ contains all the information necessary to calculate $\widehat{ETC}$ is similar to the property of sufficiency. In fact, the factorization argument used is similar to the characterization of sufficiency of Neyman (1935). However, we are not aware of the concept of sufficiency for nonparametric statistical models. Even, the definition of sufficiency relies heavily on the parametrization of the statistical model.

Thirdly, Theorem (3) can also be proven in a more direct manner, if we assume the continuity of the class conditional distributions.

Proof **of Theorem 3.** Let us consider the infimum of (4.2)

$$\inf_{t\in\mathbb{R}} \widehat{EPE}\left(\delta_{<t}\right) = \inf_{t\in\mathbb{R}} (1-\pi_1)\cdot c_0 \cdot \frac{1}{n_0}\sum_{i=1}^{n_0} \mathbb{1}\left(X_i^0 < t\right) + \pi_1 \cdot c_1 \cdot \frac{1}{n_1}\sum_{j=n_0+1}^{n} \mathbb{1}\left(X_j^1 \geq t\right).$$

Let $t' := F^0(t)$. Due to the monotonicity of F^0 we can rewrite this expression as

$$\inf_{t'\in[0,1]} (1-\pi_1)\cdot c_0 \cdot \frac{1}{n_0}\sum_{i=1}^{n_0} \mathbb{1}\left(F^0(X_i^0) < t'\right) + \pi_1 \cdot c_1 \cdot \frac{1}{n_1}\sum_{j=n_0+1}^{n} \mathbb{1}\left(F^0\left(X_j^1\right) \geq t'\right) =$$

$$= \inf_{t'\in[0,1]} (1-\pi_1)\cdot c_0 \cdot \frac{1}{n_0}\sum_{i=1}^{n_0} \mathbb{1}(U_i < t') + \pi_1 \cdot c_1 \cdot \frac{1}{n_1}\sum_{j=n_0+1}^{n} \mathbb{1}\left(U_j \geq t'\right). \tag{4.7}$$

$U_i \sim U_{0,1}$ follows the uniform distribution since $P(F(X) \leq t) = P(X \leq F^{-1}(t)) = F(F^{-1}(t)) = t$. Here F^0 is the continuous cumulative distribution function of the random variables X^0 and X^1. $F^{-1}(t) := \min\{x : F^0(x) < t\}$ denotes its inverse. Since the expression (4.7) is independent of F^0 and F^1 and since the same argument holds true for $\inf\{\widehat{EPE}(\delta_{\geq t}) : t \in \mathbb{R}\}$, so is $\widehat{ETC}$.

4.3.2 The Algorithm

A byproduct of the proof of Theorem (3) is that it allows us to shift the problem of calculating the ND of $\widehat{ETC}$ from the sample space $(\mathcal{X} \times \{0,1\})^n$ to $\mathcal{P}_{n_1,n_0}$, where we know that the distribution is uniform. Thus, we can calculate the probability of any event by simply counting the number of favorable permutations and dividing by the number of possible permutations $\binom{n}{n_0}$. This insight is the basis for the algorithm introduced in this section.

Let us define the following partition on $\mathcal{P}_{n_1,n_0}$

$$\mathcal{P}_{n_1,n_0} = \bigcup_{\substack{0 \geq fn \geq n_1 \\ 0 \geq fp \geq n_0}} S_{fn,fp}, \text{ where} \tag{4.8}$$

$$S_{fn,fp} := \{p \in \mathcal{P}_{n_1,n_0} : \phi(p) = (fn, fp)\}, \tag{4.9}$$

and $fn = 0, \ldots, n_1$ and $fp = 0, \ldots, n_0$ denote a given number of false negatives and false positives. The function $\phi : \mathcal{P}_{n_1,n_0} \to \{(fn, fp) : fn \in \{0, \ldots, n_1\} \wedge fp \in \{0, \ldots, n_0\}\}$ is required since for many permutations the optimal number of false positives and false negatives of $\widehat{ETC}_{\mathcal{P}}$ can be ambiguous since the resulting costs are the same. We, thus, need a well defined function ϕ, where $\widehat{ETC}_{\mathcal{P}}(p) = \psi \circ \phi(p)$ where $\psi(fp, fn) := c_0\pi_0 fp + c_1\pi_1 fn$. For this purpose, we need to introduce two conventions. For permutations where more than one position of the threshold yields the same minimal prediction error the threshold is set to the smallest index number. Secondly, for permutations where mapping positives to the left or the right of the threshold yields the same $\widehat{EPE}$, assigning the positives to the left is chosen.

$$\phi(p) := \begin{cases} \left(\sum\limits_{j=i}^{n} p_j, \sum\limits_{j=1}^{i-1}(1-p_j)\right), i = \min \operatorname*{argmin}\limits_{j=1,\ldots,n} E_{<j}(p) & \text{if } \min\limits_{j=1,\ldots,n} E_{<j}(p) \leq \min\limits_{j=1,\ldots,n} E_{\geq j}(p) \\ \left(\sum\limits_{j=1}^{i-1} p_j, \sum\limits_{j=i}^{n}(1-p_j)\right), i = \min \operatorname*{argmin}\limits_{j=1,\ldots,n} E_{\geq j}(p) & \text{else} \end{cases}$$

$$E_{<i}(p) := c_0\pi_0 \frac{1}{n_0} \sum_{j=1}^{i-1}(1-p_j) + c_1\pi_1 \frac{1}{n_1} \sum_{j=i}^{n} p_j$$

$$E_{\geq i}(p) := c_0\pi_0\frac{1}{n_0}\sum_{j=i}^{n}(1-p_j) + c_1\pi_1\frac{1}{n_1}\sum_{j=1}^{i-1}p_j$$

The two conventions are set implicitly in the definition of ϕ. Convention 1 is stated by using the $\min\operatorname{argmin}_i$, convention 2 is ensured by the "$\leq$" of the if statement.

With the definitions above we can denote the discrete ND under the assumptions made in Theorem 3 as

$$P\left[\widehat{ETC} = c\right] = \sum_{(fn,fp):c_0\pi_0 fp + c_1\pi_1 fn = c} \frac{|S_{fn,fp}|}{\binom{n}{n_0}}. \tag{4.10}$$

The ratios of (4.10) can be interpreted as the number of favorable events divided by the number of possible events, as in every Laplace space.

Let us further split the sets defined in (4.9), by discriminating between those permutations with the positive domain on the left and those with the positive domain on the right side of the threshold,

$$S_{fn,fp} = S^{\leftarrow}_{fn,fp} \cup S^{\rightarrow}_{fn,fp}. \tag{4.11}$$

For every permutation $p \in \mathcal{P}_{n_1,n_0}$ the position of the threshold is unambiguous by the conventions set in (4.3.2) and divides the permutation into a positive and a negative domain with $tp + fp$ and $tn + fn$ instances, respectively. The number of favorable permutations can be counted separately for each domain, since every combination of a favorable permutation of the positive domain $p^+ \in \mathcal{P}_{tp,fp}$ and a favorable permutation of the negative domain $p^- \in \mathcal{P}_{fn,tn}$ forms a valid permutation of the respective set $p = (p^+, p^-) \in S^{\leftarrow}_{fn,fp}$. Thus, the cardinality of the sets is given by the product of the number of favorable permutations of the positive domain and the negative domain.

$$|S^{\leftarrow}_{fn,fp}| = |S^{+,\leftarrow}_{fn,fp}| \cdot |S^{-,\leftarrow}_{fn,fp}| \tag{4.12}$$
$$|S^{\rightarrow}_{fn,fp}| = |S^{+,\rightarrow}_{fn,fp}| \cdot |S^{-,\rightarrow}_{fn,fp}|, \tag{4.13}$$

where

$$S^{+,\leftarrow}_{fn,fp} := \left\{p \in S^{\leftarrow}_{fn,fp} : \min_{i=1,\dots,n} E_{<i}(p) \leq \min_{i=1,\dots,n} E_{\geq i}(p)\right\}$$

and

$$S^{+,\rightarrow}_{fn,fp} := \left\{p \in S^{\leftarrow}_{fn,fp} : \min_{i=1,\dots,n} E_{<i}(p) > \min_{i=1,\dots,n} E_{\geq i}(p)\right\}.$$

The sets $S^{+,\rightarrow}_{fn,fp}$ and $S^{-,\rightarrow}_{fn,fp}$ are defined in an analogous way.

The independence of the two domains can easily be understood with Figure 4.4. Without loss of generality, consider a permutation with a positive domain on the left side. In this situation the threshold (in black) is on its optimal position. A negative instance is shifted one position to the left, from position i to $i-1$. This affects the cost of the left and the right threshold between the two instances, $\delta_{<i}$ and $\delta_{\geq i}$. All other thresholds remain unaffected. Only for $\delta_{\geq i}$ (in grey) the cost decreases by the shift. Thus, if we account for this, the current threshold remains optimal. The same holds true for instances on the right side. A shift of a positive instance away from the threshold will leave the cost of currently optimal threshold unaffected. Thus, we can modify the permutations on the left and right side independently and the overall number of permutations is given by its product.

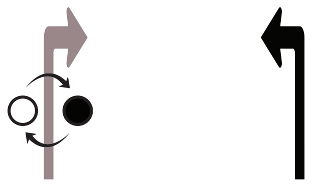


Figure 4.4: Illustration of shifting a negative instance in the positive domain. The only threshold which is affected by a decreasing cost is indicated in gray. We can, thus, shift instances away from the currently optimal threshold (in black) as long as the cost of the gray threshold exceeds the cost of the black one.

For every set of permutations and for both domains, we can construct a starting permutation, which is characterized by the fact that the false negative and false positive instances are as close to the threshold as possible. Any shift of a false instance towards the threshold will render the position of the threshold suboptimal and the resulting permutation will not be element of the respective set. From this initial permutation, the false positive and false negative instances are shifted away from the threshold as long as the obtained permutation remains in the set and the stopping permutation is reached. An example of a starting and stopping permutation for $S^{\leftarrow}_{(1,2)}$ is illustrated in Figure 4.5.

In order to properly describe this algorithm we need to introduce two conventions. The first convention concerns the indexation of the positions of a permutation. As illustrated in Figure 4.5, the indices are in increasing order starting at the threshold. The second convention concerns the indexation of the false positives or false negatives. The first false instance will always have the highest position number, the second false instance the second highest and so on. These two conventions allow us to express the number of favorable permutations of the positive domain left of the threshold by the following formula

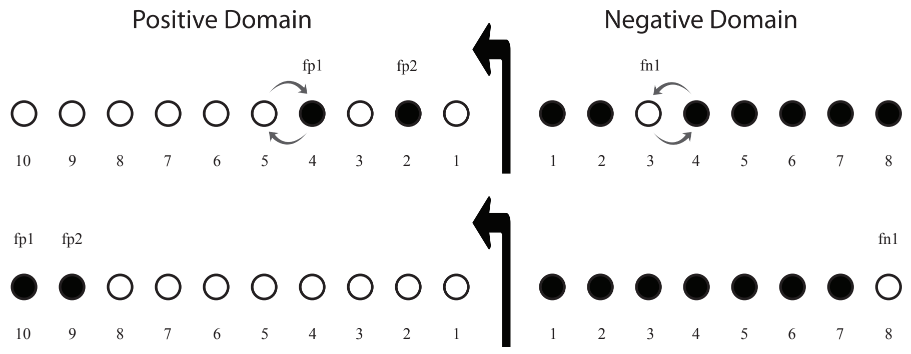


Figure 4.5: Illustration of the starting and stopping permutations for both the positive and the negative domain for $S^{\leftarrow}_{(1,2)}$ and for the parameters $n_0 = 9$, $n_1 = 9$, $c_1 = 2$, $c_0 = 1$, and $\pi_1 = 0.5$. Black circles indicate negative instances, white circles positive instances. The bold arrow points in direction of the positive domain. The numbers indicate the position indices of the algorithm according to (4.14). Inserting these parameters into (4.15) and (4.16) delivers the starting and stopping indices for the positive domain of $S_{(1,2)}$. The two false positive instances can shift from position 4 to 10 and 2 to 9, respectively. According to (4.15) and (4.16) this yields $\sum_{i_1=4}^{10}\sum_{i_2=2}^{\min\{9,i_1-1\}} 1 = 2+3+4+5+6+7+8 = 35$. As stated in (4.18) and (4.19) the false negative instance can shift from position 3 to position 8 in the negative domain, yielding $\sum_3^8 1 = 6$ permutations. According to (4.12) the overall number of permutations is, thus, $6 \cdot 35 = 210$.

$$|S^{+,\leftarrow}_{fn,fp}| = \sum_{i_1=start_1}^{stop_1} \sum_{i_2=start_2}^{\min\{stop_2,i_1-1\}} \cdots \sum_{i_{fp}=start_{fp}}^{\min\left\{stop_{fp},i_{fp-1}-1\right\}} 1, \text{ where} \tag{4.14}$$

$$start_k = \min\{tp + fp - (k-1), v_k + (fp - k) + 1\}, \tag{4.15}$$

$$v_k = \min\{n \in \mathbb{N} : nc_1\pi_1 > (fp - k + 1)c_0\pi_0\},$$

$$stop_k = \min\{tp + fp - (k-1), start_k + w_k\}, \tag{4.16}$$

$$w_k = \max\{n \in \mathbb{N} : fp\ c_0\pi_0 + fn\ c_1\pi_1 \le (tn + fp - k)c_0\pi_0 + (tp - v_k - n)\,c_1\pi_1\}.$$

The parameter $start_k$ in (4.15) denotes the starting position of the k-th false positive. For the threshold to be optimal there must be at least $\min\{n \in \mathbb{N} : nc_1\pi_1 > kc_0\pi_0\}$ positives to the right plus another $fp - k$ negatives. The index of the k-th false positive is, however, limited to $tp - fp - (k-1)$, which is the case when no positive instances are to the left.

The parameter $stop_k$ in (4.16) denotes the final stopping position of the k-th false positive. It equals the starting index plus l_2, which denotes the maximal number of shifts the positive instance can undertake before the threshold becomes suboptimal. In this case the positive domain would switch to the other side and its position would switch to

$start + l_2 + 1$. However, the stopping index can never exceed $tp + fp - (k-1)$ because at this position there are no more positives to the left of the k-th false positive.

Equivalent recursive expressions for $|S^{+,\rightarrow}_{fn,fp}|$, $|S^{-,\leftarrow}_{fn,fp}|$, and $|S^{-,\rightarrow}_{fn,fp}|$ are (4.17), (4.20), and (4.22).

$$|S^{-,\leftarrow}_{fn,fp}| = \sum_{j_1=start_1}^{stop_1} \sum_{j_2=start_2}^{\min\{stop_2, j_1-1\}} \cdots \sum_{j_{fn}=start_{fn}}^{\min\{stop_{fn}, j_{fn-1}-1\}} 1, \text{ where} \tag{4.17}$$

$$\begin{aligned} start_k &= \min\{tn + fn - (k-1), v_k + (fn-k) + 1\}, &(4.18)\\ v_k &= \{n \in \mathbb{N} : nc_0\pi_0 \geq (fn-k+1)c_1\pi_1\}, \\ stop_k &= \min\{tn + fn - (k-1), start_k + w_k\}, &(4.19)\\ w_k &= \max\{n \in \mathbb{N} : fp\ c_0\pi_0 + fn\ c_1\pi_1 < (tn - v_k - n)\, c_0\pi_0 + (tp + fn - k)c_1\pi_1\}. \end{aligned}$$

$$|S^{+,\rightarrow}_{fn,fp}| = \sum_{i_1=start_1}^{stop_1} \sum_{i_2=start_2}^{\min\{stop_2, i_1-1\}} \cdots \sum_{i_{fp}=start_{fp}}^{\min\{stop_{fp}, i_{fp-1}-1\}} 1, \text{ where} \tag{4.20}$$

$$\begin{aligned} start_k &= \min\{tp + fp - (k-1), v_k + (fp-k) + 1\}, \\ v_k &= \min\{n \in \mathbb{N} : nc_1\pi_1 \geq (fp-k+1)c_0\pi_0\}, \\ stop_k &= \min\{tp + fp - (k-1), start_k + w_k\}, &(4.21)\\ w_k &= \max\{n \in \mathbb{N} : fp\ c_0\pi_0 + fn\ c_1\pi_1 < (tp - v_k - n)\, c_1\pi_1 + (tn + fp - k)c_0\pi_0\}. \end{aligned}$$

$$|S^{-,\rightarrow}_{fn,fp}| = \sum_{j_1=start_1}^{stop_1} \sum_{j_2=start_2}^{\min\{stop_2, j_1-1\}} \cdots \sum_{j_{fn}=start_{fn}}^{\min\{stop_{fn}, j_{fn-1}-1\}} 1, \text{ where} \tag{4.22}$$

$$\begin{aligned} start_k &= \min\{tn + fn - (k-1), v_k + (fn-k) + 1\}, \\ v_k &= \min\{n \in \mathbb{N} : nc_0\pi_0 > (fn-k+1)c_1\pi_1\}, \\ stop_k &= \min\{tn + fn - (k-1), start_k + w_k\}, &(4.23)\\ w_k &= \max\{n \in \mathbb{N} : fp\ c_0\pi_0 + fn\ c_1\pi_1 \leq (tn - v_k - n)\, c_0\pi_0 + (tp + fn - k)c_1\pi_1\}. \end{aligned}$$

Consider a permutation with a threshold at the optimal position. Without loss of generality the positive domain is on the left side. In this situation we will modify the permutation by shifting a positive instance to a higher index number. To be more precise, we will shift the position of the k-th false positive from position i to $i+1$. Only the cost of thresholds located at position i are affected by such a shift. The cost of the threshold with the positive domain on the left is increased, while the cost of the threshold with the positive domain on the right side is decreased. Thus, the permutations obtained by such a shift are only valid permutations of the current set, as long as the cost of this threshold is not lower than the one of the currently optimal threshold. This is guaranteed by the inequalities (4.16), (4.19), (4.21), and (4.23).

4.3.3 Implementation

Since the starting and stopping indices of the sum of index i_k depend on the current value of the index i_{k-1} and the number of sums depends on the number fp and fn, Equation (4.14) is implemented as a recursive scheme. The initial recursion level equals the number of false positives or false negatives. The recursive function calls itself, recalculating the start and stop indices for the lower level until the base case is reached. In this case, the algorithm simply returns $stop_{i_{fp}} - start_{i_{fp}}$. A flow chart of this scheme can be found in Figure 4.6.

The strength of this recursive scheme is that it only requires three arguments: level, start, and stop. This means that, except for the parameters, the number of possible positions of i_k only depends on the position of i_{k-1} and i_{k+1}. The position of the other instances are irrelevant.

```
recursive_function(map, level, start, stop) {
  if (level, start, stop) in map.keys {
    return map.values[(level,start,stop)]
  } else {
    if level==1 {
      map.value[(level,start,stop)] = stop - start
      return stop - start
    } else {
      sum = 0;
      for i in start to stop {
        start.new = calc.start(i, fp, fn, start, stop)
        stop.new = calc.stop(i, fp, fn, start, stop)
        sum = sum + recursive_function(map, level - 1, start.new, stop.new)
      }
     map.value[(level, start, stop)] = sum
    return sum
    }
  }
}
```

The correctness of the implemented algorithm was tested in two different ways. For sample sizes up to 15, it was computationally feasible to generate a matrix with all possible permutations using the `allPerm` function from the `multicool` package. The algorithm is described in Williams (2009). Then row-wise the function *phi* was applied and different statistics such as the overall number of permutations for every pair (fp, fn), the distribution of the cost values, etc. was calculated. The second approach, which allows to test the correct working of the algorithm for larger sample sizes was simply to add up the number of permutations for every point in the support and to verify that they add up to $\binom{n}{n_1}$. The code for these tests can be found in `etc_tests.R`.

The described method has been implemented for the R software platform (R Core

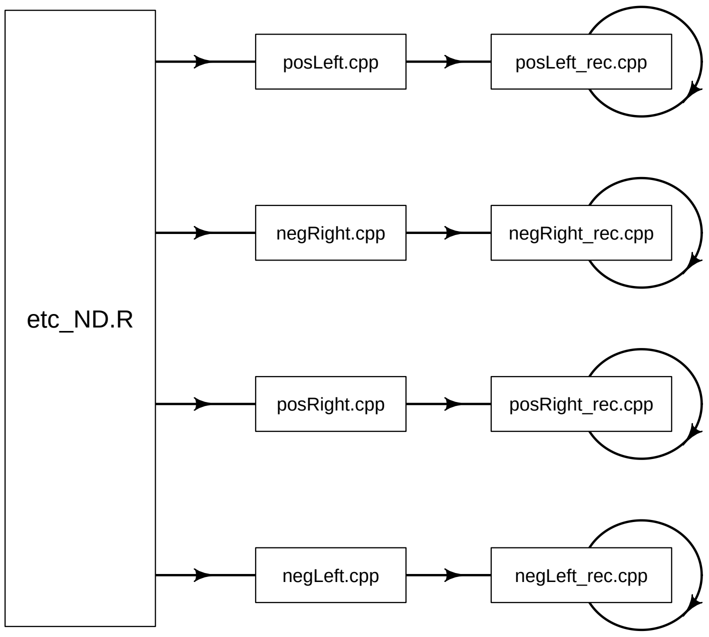


Figure 4.6: This figure illustrates the structure of functions used for the calculation of the null distribution of $\widehat{ETC}$. Different functions are required for the calculation of the favorable permutation of the positive domain on the left and right and the negative domain on the left and right. These functions further call recursive functions, calling themselves until the base case is reached.

Team, 2015) and is part of the `UNIC` package that has been published unter the GNU general public license. The challenges of implementing it were twofold. First, since it is recursive in nature, the number of function calls grows exponentially as the sample size increases and so does the required computation time, which is $O(2^n)$, where n denotes the sample size. Especially at lower recursion levels the function will be called repeatedly with the exact same arguments. This can be avoided by means of memoization or dynamic programming techniques which store the input arguments and the output in a suitable data object and use the cached results instead of calling the function again. Furthermore, the recursive functions were implemented in C++ using the `Rcpp` package, see Eddelbuettel and François (2011). The modified algorithm is of the order $O(1.05^n)$. The performance gains resulting from these modifications are illustrated in Figure 4.7.

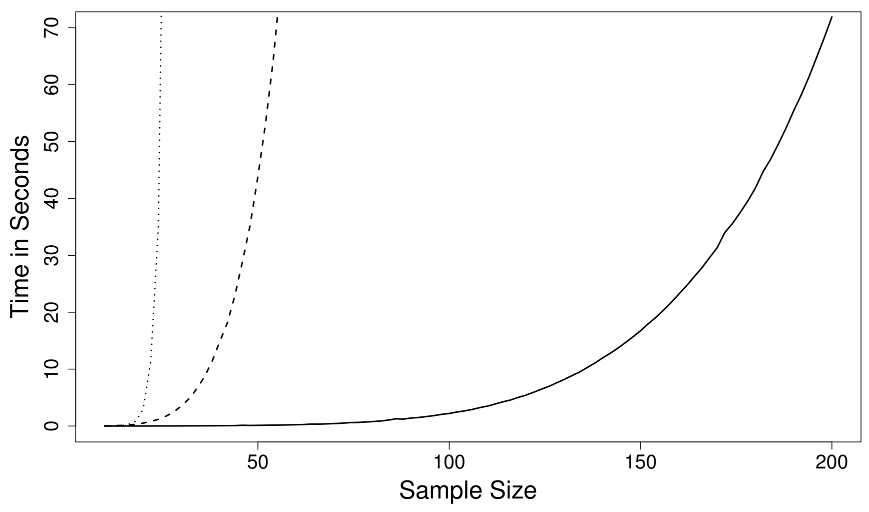


Figure 4.7: This figure illustrates the time necessary for computing the ND on a standard PC with 4 kernels in seconds as a function of the sample size. The dotted line represents the standard algorithm implemented in R. The dashed line introduces memoization to the algorithm within the R platform. The solid line, which clearly, outperforms the others, represents the algorithm implemented in C++ including memoization.

Secondly, as the sample size increases, the number of possible permutations quickly exceeds the maximum integer value. This happens when the sample contains more than 30 instances. Thus, high precision numbers are needed to run the algorithm and save its results. The GNU multiple precision arithmetic library GMP (Granlund et al., 1993–2017) was used.

Another challenge concerns the existence of ties in the data. Theoretically, if the distribution of X is continuous, there will almost surely be no tied order statistics. However, in real data applications, there might be instances with the same value and different class labels. The implemented algorithm proceeds by calculation of both test statistics and returning the more conservative value.

CHAPTER 5

EIC: Expected Prediction Error of the Interval Classifier

In this section a non-parametric equivalent of EBC with unequal variances ($\sigma_0 \neq \sigma_1$) will be introduced and studied. In Theorem 1 we have learned that for Gaussian class conditionals with unequal variances the optimal classifier is an interval classifier. In this section we will chose a more direct nonparametric approach to deriving a classifier. Instead of making a parametric assumption for the CCDs and fitting the classifier by optimizing the parameters, we will limit our search to the family of interval classifiers, defined in (5) and minimize the sample error. Let us define EIC, the expected prediction error of the optimal interval classifier

$$EIC := \min_{\delta \in \mathcal{F}_I} EPE(\delta). \tag{5.1}$$

This statistic estimates the prediction error resulting from mapping all values from an interval to one class and all values from its complement to the other class. It ranges from 0, when perfect separation is possible, to $\min(c_0\pi_0, c_1\pi_1)$, when X exhibits no discriminatory power with respect to the classes of Y.

5.1 Sample Estimate of EIC

Again, suppose $(x_i, y_i), i = 1, \dots, n$, are i.i.d. realizations of (X, Y) and let $n_1 := \sum_{i=1}^n y_i$ and $n_0 := n - n_1$ denote the number of positives and negatives, respectively. Without loss of generality let us assume that $y_i = 0$ for $i = 1, \dots, n_0$ and $y_i = 1$ for $i = n_0 + 1, \dots, n$. To derive a sample estimate of EIC we simply need to substitute the false positive and false negative rates in (2.6) by their empirical counterparts (2.7). Note, that due to the specific form of an interval classifier its false negative rate $P[\delta_{(t_1,t_2]}(X) = 1|Y = 0]$ is simply $F^0(t_2) - F^0(t_1)$, the difference of the cumulative CCD evaluated at t_2 and t_1.

By the same argument $P[\delta_{\mathbb{R}\setminus(t_1,t_2]}(X) = 0|Y = 1] = 1 - (F^1(t_2) - F^1(t_1))$. Substituting these expressions by their empirical counterparts, yields

$$\begin{aligned}
\widehat{EPE}(\delta_{(t_1,t_2]}) &= \underbrace{c_0\pi_0 \frac{1}{n_0}\sum_{i:y_i=0} \mathbb{1}_{(t_1,t_2]}(x_i)}_{\text{false positive rate}} + \underbrace{c_1\pi_1 \frac{1}{n_1}\sum_{i:y_i=1} \mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(x_i)}_{\text{false negative rate}} \\
\widehat{EPE}(\delta_{\mathbb{R}\setminus(t_1,t_2]}) &= \underbrace{c_0\pi_0 \frac{1}{n_0}\sum_{i:y_i=0} \mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(x_i)}_{\text{false positive rate}} + \underbrace{c_1\pi_1 \frac{1}{n_1}\sum_{i:y_i=1} \mathbb{1}_{(t_1,t_2]}(x_i)}_{\text{false negative rate}} .
\end{aligned} \tag{5.2}$$

Note that $\widehat{EPE}(\delta_{(t_1,t_2]})$ is constant e.g. for all thresholds $t_1 \in [x_{(i)}, x_{(i+1)})$, where $x_{(i)}$ denotes the j-th order statistic of the sample and the same holds true for all thresholds $t_2 \in [x_{(j)}, x_{(j+1)})$. Thus, instead of having to search $\mathbb{R}$ it suffices to find the minimum of $(n+1)^2/2$ values. Exploiting this fact leads us to the empirical counterpart of (5.1),

$$\widehat{EIC} = \min\left\{\min_{i\le j=1,\dots,n} \widehat{EPE}(\delta_{(x_i,x_j]}), \min_{i\le j=1,\dots,n} \widehat{EPE}(\delta_{\mathbb{R}\setminus(x_i,x_j]})\right\}. \tag{5.3}$$

The method can more easily be understood when it is described in an algorithmic manner.

EIC Consider the data $(x_i, y_i)_{i=1,\dots,n}$.

- Sort $(x_i, y_i), i = 1, \dots, n$ in increasing order, such that $x_{(1)} \le x_{(2)} \le \dots \le x_{(n)}$.
- Calculate $\widehat{\text{EPE}}(\delta_{(x_{(i)},x_{(j)}]})$ and $\widehat{\text{EPE}}(\delta_{\mathbb{R}\setminus(x_{(i)},x_{(j)}]})$ for $i \le j = 1, \dots, n$ and
- find $\widehat{EIC} = \min\left\{\widehat{\text{EPE}}(\delta_{(x_{(i)},x_{(j)}]}), \widehat{\text{EPE}}(\delta_{\mathbb{R}\setminus(x_{(i)},x_{(j)}]}) : i \le j = 1, \dots, n\right\}$.

Note that the sample estimate of $\widehat{EIC}$ will always be lower than the sample estimate of $\widehat{ETC}$. This becomes clear when one realizes that the result of the optimal threshold classifier can be obtained by setting one boundary to the position of the threshold and the second either above the maximum sample value or below the minimum sample value depending on the direction of the threshold classifier.

5.2 Properties of $\widehat{EIC}$

This chapter will establish some results on the bias, consistency and efficiency of the sample estimate of EIC. Following the same arguments as in Section 4.2 we will see that

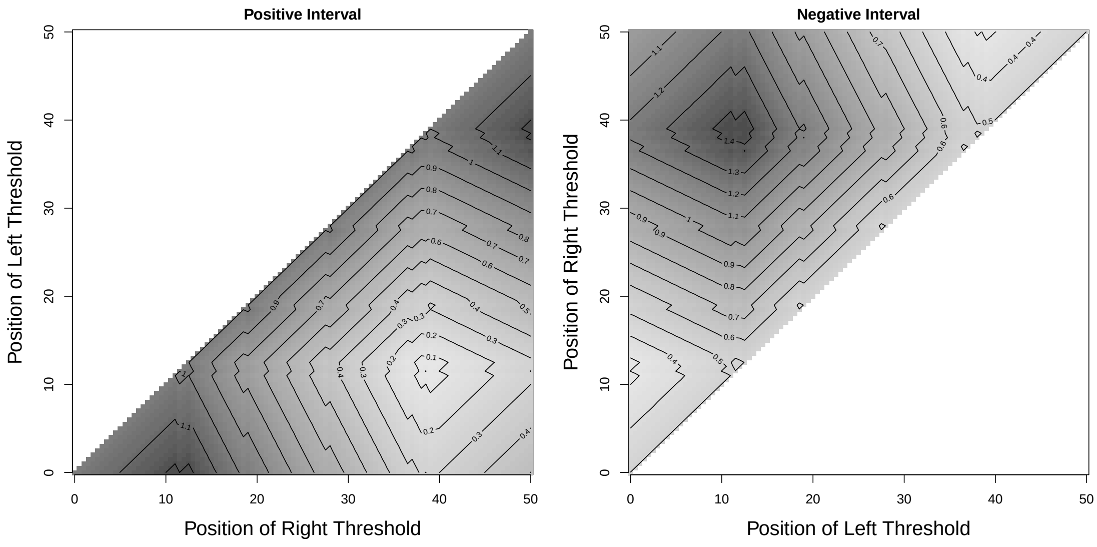


Figure 5.1: Illustration of the EIC algorithm. The plot on the left illustrates the cost generated by an interval classifier with a positive interval depending on the position of the left and right boundaries. The x and y-axis denote the rank in the vector of ordered values. The right plot illustrates the costs of a negative interval.

$\widehat{EIC}$ is a biased estimator of EIC even though, $\widehat{EPE}(\delta_{(t_1,t_2]})$ is an unbiased estimator of $EPE(\delta_{(t_1,t_2]})$,

$$\begin{aligned}\mathbf{E}\left[\widehat{EPE}(\delta_{(t_1,t_2]})\right] &= c_0\pi_0\tfrac{1}{n_0}\textstyle\sum_{i:y_i=0}\mathbf{E}\mathbb{1}_{(t_1,t_2]}(x_i) + \; c_1\pi_1\tfrac{1}{n_1}\textstyle\sum_{i:y_i=1}\mathbf{E}\mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(x_i) = \\ &= c_0\pi_0(F^0(t_2)-F^0(t_1)) + c_1\pi_1\left(1-(F^1(t_2)-F^1(t_1))\right),\end{aligned}$$

since $\mathbb{1}_{(t_1,t_2]}(X^0) \sim \mathcal{B}_{1,F^0(t_2)-F^0(t_1)}$ and $\mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(X^0) \sim \mathcal{B}_{1,1-(F^0(t_2)-F^0(t_1)}$.

Let us assume without loss of generality that $\delta_{(t_1^*,t_2^*]}$ yields the minimal EPE among all interval classifiers. Trying to estimate t_1^* and t_2^* by means of the samples x_i and x_j which minimize $\widehat{EPE}(\delta_{(x_i,x_j]})$ introduces a bias. Since there will always exist indices $i \leq j \in 1,\ldots,n$ such that $\widehat{EPE}(\delta_{(x_i,x_j]}) = \widehat{EPE}(\delta_{(t_1^*,t_2^*]})$, the following inequality holds

$$\mathbf{E}\left[\min_{i\leq j=1,\ldots,n}\widehat{EPE}(\delta_{(x_i,x_j]})\right] \leq \mathbf{E}\left[\widehat{EPE}(\delta_{(t_1^*,t_2^*]})\right].$$

In order to visualize the bias of $\widehat{EIC}$ a simulation study was conducted. Random numbers of two Gaussian variables with an increasing difference in their means and an increasing ratio of their variances were drawn repeatedly and the mean values of the statistics of $\widehat{EIC}$ and $\widehat{EBC}$ were calculated. The true prediction error of the Bayes classifier was calculated and is depicted in Figure 5.2. The differences to the true prediction error are

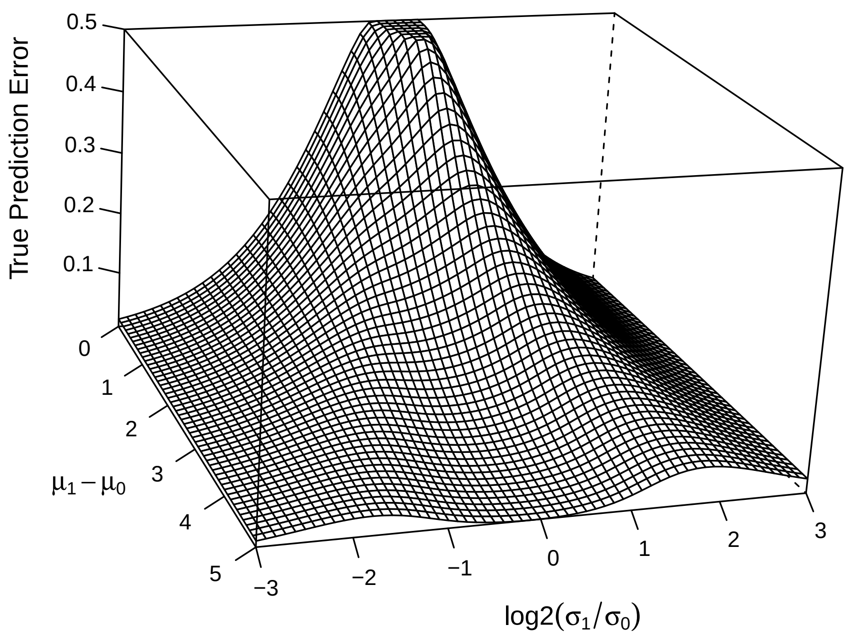
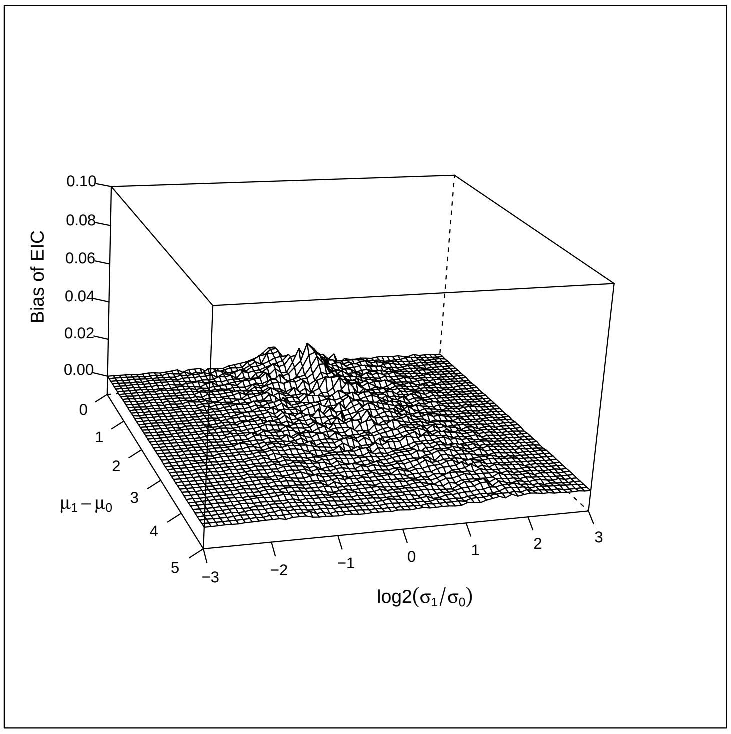


Figure 5.2: This figure illustrates the true prediction error of the Bayes Classifier for two Gaussian class conditionals $X^0 \sim N(\mu_0, \sigma_0^2)$ and $X^1 \sim N(\mu_1, \sigma_1^2)$ as a function of $\mu_1 - \mu_0$ and $\log 2(\sigma_1/\sigma_0)$. One can clearly see that as the CCDs become less alike either by an increasing difference in their central location or an increasing difference between their variances the expected error of a prediction falls to 0.

depicted in Figure 5.3. It is not surprising that $\widehat{EBC}$ shows only a very small bias, which probably stems from the estimates of the mean and the variance, which are itself no unbiased estimates of the parameters of the Gaussian distribution. $\widehat{EIC}$, however, shows a significant bias which increases as the discriminatory power of the random variable decreases.



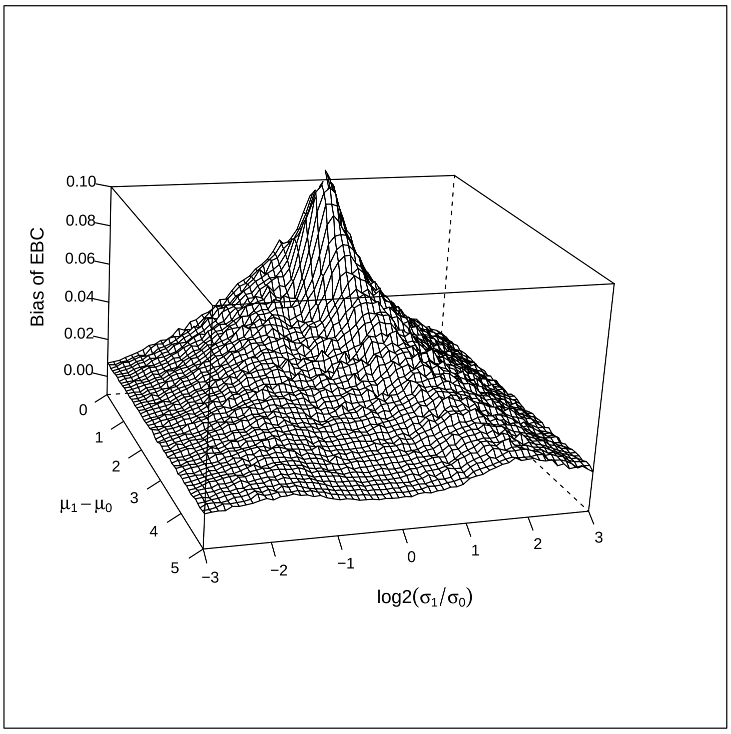


Figure 5.3: The figures illustrate the bias of the estimates $\widehat{EIC}$ and $\widehat{EBC}$ as a function of $\mu_1 - \mu_0$ and $\log 2(\sigma_1/\sigma_0)$. The bias, thus, depends strongly on the discrimatory power of the class conditionals. For Gaussian class conditionals with a limited discriminatory power the bias is greatest.

The following theorem will establish the consistency of $\widehat{ETC}$. It is more or less a corollary of theorem 2 since the arguments are exactly the same.

Theorem 4 *Assume continuous class conditional distributions F^0 and F^1 and a unique minimizer of EIC, then $\widehat{EIC}$ is a consistent estimator of EIC,*

$$plim_{n\to\infty}\widehat{EIC}_n = EIC.$$

Proof 4 *Consider an independent and identically distributed sample $(X_i, Y_i)_{i=1,\dots,n}$, where*
$X_i|Y_i = 0 \sim F^0$ and $X_i|Y_i = 1 \sim F^1$. To establish the consistency, we have to show that

$$\forall\epsilon > 0 : \lim_{n\to\infty} P\left[\left|\widehat{EIC}_n - EIC\right| > \epsilon\right] = 0. \tag{5.4}$$

Without loss of generality assume that $\delta_{(t_1^*,t_2^*]}$ minimizes the prediction error among all interval classifiers

$$\delta_{(t_1^*,t_2^*]} = \operatorname*{argmin}_{\delta\in\mathcal{F}_I} EPE(\delta)$$

and, thus,

$$EIC = \min_{\delta\in\mathcal{F}_I} EPE(\delta) = EPE(\delta_{(t_1^*,t_2^*]}) = c_0\pi_0(F^0(t_2^*)-F^0(t_1^*))+c_1\pi_1(1-(F^1(t_2^*)-F^1(t_1^*))).$$

Inserting this expression is (5.4) *yields*

$$\forall\epsilon > 0 : \lim_{n\to\infty} P\left[\left|\widehat{EIC}_n - \Big(c_0\pi_0(F^0(t_2^*) - F^0(t_1^*)) + c_1\pi_1(1 - (F^1(t_2^*) - F^1(t_1^*)))\Big)\right| > \epsilon\right] = 0.$$

Lets us start with the definition of $\widehat{EIC}$, perceived as a random variable

$$\widehat{EIC}_n = \min\left\{\min_{i\le j=1,\dots,n}\widehat{EPE}(\delta_{(X_i,X_j]}), \min_{j=1,\dots,n}\widehat{EPE}(\delta_{\mathbb{R}\setminus(X_i,X_j]})\right\}.$$

The minimum is a continuous function and, thus, by virtue of the continuous mapping theorem all we need to show is the consistency of its arguments. Due to the assumptions made about $\delta_{(t_1^*,t_2^*]}$ we will show the convergence of the first argument. As $n \to \infty$, both $n_0, n_1 \to \infty$ at rates $n_1 \approx \pi_1 n$ and $n_0 \approx \pi_0 n$. Further, let us denote $\hat{F}^0_{n_0}(x) := \frac{1}{n_0}\sum_{i:Y_i=0}\mathbb{1}_{(-\infty,x)}(X_i)$ and $\hat{F}^1_{n_1}(x) := \frac{1}{n_1}\sum_{i:Y_i=1}\mathbb{1}_{(-\infty,x)}(X_i)$. The problem then translates to showing that $\forall\epsilon > 0$:

$$\lim_{n\to\infty} P\Bigg[\Bigg|\min_{i\le j=1,\dots,n} c_0\pi_0\left(\hat{F}^0_{n_0}(X_j) - \hat{F}^0_{n_0}(X_i)\right) + c_1\pi_1\left(1 - \left(\hat{F}^1_{n_1}(X_j) - \hat{F}^1_{n_1}(X_i)\right)\right) - \\ - \Big(c_0\pi_0(F^0(t_2^*) - F^0(t_1^*)) + c_1\pi_1(1 - (F^1(t_2^*) - F^1(t_1^*)))\Big)\Bigg| > \epsilon\Bigg] = 0$$

By virtue of the Glivenko-Cantelli theorem, the empirical distribution functions converge almost surely uniformly to the true distribution functions $\sup_{x\in\mathbb{R}} |\hat{F}^0_{n_0}(x) - F^0(x)| \xrightarrow{a.s.} 0$ *and* $\sup_{x\in\mathbb{R}} |\hat{F}^1_{n_1}(x) - F^1(x)| \xrightarrow{a.s.} 0$. *If we define*

$$Q_n(t_1, t_2) := c_0\pi_0 \left(\hat{F}^0_{n_0}(t_2) - \hat{F}^0_{n_0}(t_1)\right) + c_1\pi_1 \left(1 - \left(\hat{F}^1_{n_1}(t_2) - \hat{F}^1_{n_1}(t_1)\right)\right)$$

and

$$Q(t_1, t_2) := \Big(c_0\pi_0(F^0(t_2) - F^0(t_1)) + c_1\pi_1(1 - (F^1(t_2) - F^1(t_1)))\Big)$$

then

$$\sup_{(t_1,t_2)\in\mathbb{R}^2} \Big|Q_n(t_1, t_2) - Q(t_1, t_2)\Big| \xrightarrow{a.s.} 0,$$

will also converge uniformly almost surely.

Note that since F^0 *and* F^1 *are continuous, so is* Q, *which is a nonrandom function with a unique minimum at* $\theta^* = (t^*_1, t^*_2)$. *Further, let us denote the minimizer of* Q_n *as*

$$\hat{\theta}_n := \arg \min_{(t_1,t_2)\in\mathbb{R}^2} Q_n(t_1, t_2) = \min_{i\leq j=1,\ldots,n} Q_n(X_i, X_j).$$

We will now argue that under these circumstances the minimizer of Q_n *will converge in probability to the minimizer of* Q.

For every $\epsilon > 0$ *we have*

$$c(\epsilon) := \inf_{\theta\in\Theta:||\theta-\theta^*||\geq\epsilon} Q(\theta) > Q(\theta^*)$$

since $\{\theta \in \Theta \subseteq \mathbb{R}^2 : ||\theta - \theta^*|| \geq \epsilon\}$ *is compact and* $Q(\theta)$ *is continuous, and since* θ^* *is the unique minimizer of* Q. *Choose* $0 < \delta < 1/2(c(\epsilon) - Q(\theta^*))$, *i.e.,* δ *is such that*

$$c(\epsilon) - \delta > Q(\theta^*) + \delta.$$

Note that $\delta = \delta(\epsilon)$. *On the event*

$$\left\{\sup_{\theta\in\Theta} |Q_n(\theta) - Q(\theta)| < \delta\right\}$$

we then have

$$\inf_{\theta\in\mathbb{R}^2:||\theta-\theta^*||\geq\epsilon} Q_n(\theta) \geq \inf_{\theta\in\Theta:||\theta-\theta^*||\geq\epsilon} Q(\theta) - \delta = c(\epsilon) - \delta > Q(\theta^*) + \delta \geq Q_n(\theta^*) \geq Q_n(\hat{\theta}_n)$$

That is, on the above event we obtain that

$$||\hat{\theta}_n - \theta^*|| < \epsilon.$$

In other words,

$$\left\{||\hat{\theta}_n - \theta^*|| < \epsilon\right\} \supseteq \left\{\sup_{\theta \in \Theta} |Q_n(\theta) - Q(\theta)| < \delta\right\}.$$

But

$$P\left(\sup_{\theta \in \Theta} |Q_n(\theta) - Q(\theta)| < \delta\right) \to 1,$$

and hence

$$P\left(||\hat{\theta}_n - \theta^*|| < \epsilon\right) \to 1$$

which establishes the required consistency.

□

5.3 Derivation of the Null Distribution

In this section we will establish the distribution of the test statistic $\widehat{EIC}$ under the null hypothesis $H_0 : F^0 = F^1$. This distribution allows to assign a p-value to every value of $\widehat{EIC}$. It, thus, yields the probability that the observed or a more extreme outcome is obtained under the null hypothesis that X exhibits no information about the class membership.

5.3.1 Independence of the CCDs under H_0

Similar to the theorem in Section 4.3.1, we would like to obtain the result that the ND of $\widehat{EIC}$ is independent of F^0 and F^1 under $H_0 : F^0 = F^1$. The proof follows the exact same arguments as Theorem 3.

Theorem 5 *Consider the i.i.d. samples of the class-conditional random variables $X_i^0 \sim F^0, i = 1, \ldots, n_0$ and $X_i^1 \sim F^1, i = n_0 + 1, \ldots, n$. Under the null hypothesis $H_0 : F^0(x) = F^1(x), \forall x \in \mathcal{X}$ the sampling distribution of $\widehat{EIC}$ is independent of F^0 and F^1.*

Proof 5 *Consider r, defined in* (4.5)*, which will factorize $\widehat{EIC}$ in the following manner $\widehat{EIC} \equiv \widehat{EIC}_{\mathcal{P}} \circ r$. The following commutative diagram illustrates this factorization.*

$$\begin{tikzcd}
(\mathcal{X}, \{0,1\})^n \arrow[rr, "\widehat{EIC}"] \arrow[dr, "r"'] & & \mathbb{R}^+ \\
& \mathcal{P}_{n_1,n_0} \arrow[ur, "\widehat{EIC}_{\mathcal{P}}"'] &
\end{tikzcd}$$

If we show the validity of this factorization, all other arguments of the proof of Theorem (3) *hold true. Thus, let us rewrite the first argument of* (5.3)*.*

$$\begin{aligned}\min_{i\leq j=1,\dots,n} \widehat{EPE}(\delta_{(x_{(i)},x_{(j)}]}) &= \min_{i\leq j=1,\dots,n} c_0\pi_0\frac{1}{n_0}\sum_{i:y_i=0}\mathbb{1}_{(t_1,t_2]}(x_i) + c_1\pi_1\frac{1}{n_1}\sum_{i:y_i=1}\mathbb{1}_{\mathbb{R}\setminus(t_1,t_2]}(x_i)\\ &= \min_{i\leq j=1,\dots,n} c_0\pi_0\frac{1}{n_0}\sum_{k=i}^{j}(1-y_{(k)}) + c_1\pi_1\frac{1}{n_1}\left[\sum_{k=1}^{i}y_{(k)} + \sum_{k=j+1}^{n}y_{(k)}\right]\end{aligned}$$

The factorization of $\min_{i\leq j=1,\dots,n}\widehat{EPE}(\delta_{\mathbb{R}\setminus(t_1,t_2]})$ *follows along the same lines.*

□

The derivation of the ND of $\widehat{EIC}$ differs from the approach chosen for the ND of $\widehat{ETC}$. For $\widehat{ETC}$ the calculation is based on an efficient recursive counting scheme in $\mathcal{P}_{n_1,n_0}$ which requires only three arguments, the recursion level and the starting and stopping value for the index, three integer values. Unfortunately, for $\widehat{EIC}$ an analogous approach is not entirely possible, since it would require the position of all the false instances in the domain. However, for "small values" of fp and fn, the position of the neighboring instances are sufficient. Thus, a similar recursive scheme, introduced in Section 5.3.2, can be used to calculate the ND of the tail in an exact manner. For the rest of the distribution we will use two other approaches. The region around the mode of the ND will be obtained by means of random permutations of the class labels, see Section 5.3.3. This is where the applied method yields the most accurate approximations of the true value of the ND. All other points in the support of the ND will be obtained by means of an interpolation, see Section 5.3.4. The approach is illustrated in Figure 5.5.

5.3.2 The Algorithm

The counting algorithm for $\widehat{EIC}$ proceeds in a similar manner as in Section 4.3.2. However, we will see that the simplified version presented in this section only allows an exact calculation up to $fp, fn \leq floor(\min(n_1, n_0)/2)$. An algorithm suitable for the entire support of the ND would require many more arguments rendering the memoization inefficient. Thus, the calculation would not be feasible in a reasonable time span.

First we shall shift the problem of calculating the ND of $\widehat{EIC}$ from the sample space $(\mathcal{X}, \{0,1\})^n)$ to $\mathcal{P}_{n_1,n_0}$, where we know that the induced distribution is uniform. Then we will define a partition of $\mathcal{P}_{n_1,n_0}$ with sets R that are characterized by the fact that they incur the same number of false positives and false negatives under the optimal interval classifier.

$$\mathcal{P}_{n_1,n_0} = \bigcup_{\substack{0\geq fn\geq n_1\\ 0\geq fp\geq n_0}} R_{fn,fp}, \text{ where} \tag{5.5}$$

$$R_{fn,fp} := \{p\in\mathcal{P}_{n_1,n_0} : \psi(p) = (fn,fp)\}, \tag{5.6}$$

$$\psi(p) := \begin{cases} \left(\sum_{k=1}^{i} p_k + \sum_{k=j+1}^{n} p_k, \sum_{k=i+1}^{j} 1 - p_k\right), (i,j) = \operatorname*{argmin}_{i \leq j} j - i : \operatorname*{argmin}_{i \leq j = 1,\dots,n} E_{\{i,\dots,j\}}(p) \\ \qquad \text{if } \min_{j=1,\dots,n} E_{\{i,\dots,j\}}(p) \leq \min_{j=1,\dots,n} E_{\{1,\dots,i\}\cup\{j+1,\dots,n\}}(p) \\ \left(\sum_{k=i+1}^{j} p_k, \sum_{k=1}^{i} 1 - p_k + \sum_{k=j+1}^{n} 1 - p_k\right), (i,j) = \min_{j-i} \operatorname*{argmin}_{i \leq j = 1,\dots,n} E_{\{1,\dots,i\}\cup\{j+1,\dots,n\}}(p) \\ \qquad \text{else} \end{cases} \tag{5.7}$$

Also for $\widehat{EIC}$ for a given permutation the minimal prediction error can be obtained by different positions of the thresholds. Thus, we require two conventions which are implicit in Equation (5.7). First, if there are several pairs (fp, fn) which yield the same cost, the pair with the smallest number of false negatives is chosen. This is ensured by the expression $\operatorname{argmin} j - i$. Secondly, if a positive interval yields the same cost as a positive complement, then the positive interval is selected. This convention is ensured by the "$\leq$" in the if statement of (5.7).

The algorithm devised in the section is supposed to count the permutation for the sets $R_{fn,fp}$ which lie on the tail of the ND. First, we shall further split these sets

$$R_{fn,fp} = R^{+}_{fn,fp} \cup R^{-}_{fn,fp},$$

where $R^{+}_{fn,fp}$ denotes the set of permutations with a positive interval domain

$$R^{+}_{fn,fp} := \left\{ p \in \mathcal{P}_{n_1,n_0} : \min_{i \leq j = 0,\dots,n} E_{\{i+1,\dots,j\}} \leq \min_{i \leq j = 0,\dots,n} E_{\{0,\dots,1\}\cup\{j,\dots,n\}} \right\}$$

and $R^{-}_{fn,fp}$ denotes the set of permutations with a negative interval domain

$$R^{-}_{fn,fp} := \left\{ p \in \mathcal{P}_{n_1,n_0} : \min_{i \leq j = 0,\dots,n} E_{\{i+1,\dots,j\}} > \min_{i \leq j = 0,\dots,n} E_{\{0,\dots,1\}\cup\{j,\dots,n\}} \right\}$$

For every permutation $p \in \mathcal{P}_{n_1,n_0}$ the position of the left and the right threshold is unambiguous by convention and divides the permutation into an interval and a complement with $tp + fn$ or $tn + fp$ instances, respectively, depending on the position of the positive domain. The complement consists of a left and a right side. The instances of the complement are divided between the two sides. The number of favorable permutations can be counted separately for each domain since every combination of a permutation of the left complement, the interval, and the right complement form a valid permutation. Thus, similar to Equation (4.12) we can express the cardinality of $R^{+}_{fn,fp}$ and $R^{-}_{fn,fp}$ as

$$|R^{+}_{fn,fp}| = |R^{-,\leftarrow}_{fn,fp}| \cdot |R^{+,><}_{fn,fp}| \cdot |R^{-,\rightarrow}_{fn,fp}|$$

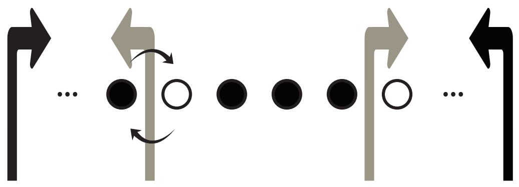


Figure 5.4: This figure illustrates two characteristics of the algorithm. First, it shows the independence of the domains. The optimality of the boundaries indicated in black after a shift of a negative instance in a positive domain only depends on the position of the negative instances in the same domain. Secondly, it also depicts a situation which illustrates why a recursive function calculating the number of permutations for a given set must take more arguments into account than the level, the start index, and the stopping index. In this case cost attributed to the gray boundaries depend on the position of the false negative instances to the right of the shifting instance. If there are three negative instances in a row, then the gray boundaries will become optimal. Thus, the recursive scheme must consider the position of every negative instance, rendering the memoization method ineffective and ultimately making the execution of the recursive schema computationally infeasible for sample sizes above 20.

$$|R^{-}_{fn,fp}| = |R^{+,\leftarrow}_{fn,fp}| \cdot |R^{-,><}_{fn,fp}| \cdot |R^{+,\rightarrow}_{fn,fp}|$$

For every set we can construct a starting permutation. For the complement this permutation is characterized by the quality that the false instances are as close to the boundaries of the interval as possible. For the interval the starting permutation is characterized by the quality that the instances are as far left as possible. The number of instances in the left complement can vary between 0 and $fp + tn$ or $fn + tp$ for $R^{+,\leftarrow}_{fn,fp}$ or $R^{-,\leftarrow}_{fn,fp}$, respectively. The same holds true for the right complements $R^{+,\rightarrow}_{fn,fp}$ and $R^{-,\rightarrow}_{fn,fp}$. Thus, we need to generate the starting and stopping permutations for every single cardinality.

There is a big difference compared to the algorithm of Section 4.3.2. Remember that the starting and stopping indexes of false instance i depend only on the current positions of $i-1$ and $i+1$, if they exist. This nice feature allows a very easy recursive algorithm. This nice feature, unfortunately, does not hold true for $\widehat{EIC}$. In Figure 5.4 we can see an example of a situation where the evaluation of whether an instance can shift one position further depends on many other instances in this domain.

In the following expression $fp.l$, $fp.r$, $fn.l$, and $fn.r$ denote the false positives and the false negatives on the left and the right complement, respectively. The indexing convention is as follows: The positions in the interval are numbered from left to right. The positions in the complement increase when moving away from the threshold. The false instances in the interval are numbered from left to right. The left-most instance has the number one which corresponds with the level of the recursive algorithm. The false instances in the complements are numbered from the extremes to the center. The outermost false instance has the number one, which again corresponds with the level of the recursive algorithm.

$$|R^{+,><}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\ldots,n\} : v_k + k > fp + tp - (fp - k - 1) - v_{fp-k+1} \\ \sum\limits_{i_1=start_1}^{\min\{i_2-1,stop_1\}} \sum\limits_{i_2=\max\{i_1+1,start_2\}}^{\min\{i_3-1,stop_2\}} \cdots \sum\limits_{i_{fp}=\max\{i_{fp-1}+1,start_{fp}\}}^{stop_{fp}} 1 & \text{else} \end{cases}, \tag{5.8}$$

where

$$\begin{aligned} v_k &= \min\{n \in \mathbb{N} : nc_1\pi_1/n_1 \geq kc_0\pi_0/n_0\}, & (5.9)\\ start_k &= v_k + fp - k + 1, & (5.10)\\ stop_k &= fp + tp - (k-1) - v_{fp-k+1}. & (5.11) \end{aligned}$$

The name of the implementation of this recursive scheme is `posInt.cpp`.

$$|R^{-,><}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\ldots,n\} : v_k + k \geq fn + tn - (fn - k - 1) - v_{fn-k+1} \\ \sum\limits_{i_1=start_1}^{\min\{i_2-1,stop_1\}} \sum\limits_{i_2=\max\{i_1+1,start_2\}}^{\min\{i_3-1,stop_2\}} \cdots \sum\limits_{i_{fn}=\max\{i_{fn-1}+1,start_{fn}\}}^{stop_{fn}} 1 & \text{else} \end{cases}, \tag{5.12}$$

where

$$\begin{aligned} v_k &= \min\{n \in \mathbb{N} : nc_0\pi_0/n_0 \geq kc_1\pi_1/n_1\}, & (5.13)\\ start_k &= v_k + fn - k + 1, & (5.14)\\ stop_k &= fn + tn - (k-1) - v_{fn-k+1}. & (5.15) \end{aligned}$$

The name of the implementation of this recursive scheme is `negInt.cpp`.

$$|R^{+,\leftarrow}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\ldots,n\} : v_k + fp.l - k + 1 > fp.l + tp.l - w_k \\ \sum\limits_{i_1=\max\{i_2+1,start_1\}}^{stop_1} \sum\limits_{i_2=\max\{i_3+1,start_2\}}^{\min\{i_1-1,stop_2\}} \cdots \sum\limits_{i_{fp.l}=start_{fp.l}}^{\min\left\{i_{fp.l-1}-1,stop_{fp.l}\right\}} 1 & \text{else} \end{cases}, \tag{5.16}$$

where

$$\begin{aligned} v_k &= \min\{n \in \mathbb{N} : nc_1\pi_1/n_1 \geq kc_0\pi_0/n_0\}, & (5.17)\\ start_k &= v_k + k, & (5.18)\\ w_k &= \min\{n \in \mathbb{N} : (n + tp.r)c_1\pi_1/n_1 \geq c_0\pi_0/n_0(fp.r + (fp.l - k + 1))\} & (5.19)\\ stop_k &= fp.l + tp.l - (w_k + (fp.l - k)). & (5.20) \end{aligned}$$

$$|R^{+,\rightarrow}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\dots,n\} : v_k + fp.r - k + 1 \geq fp.r + tp.r - w_k \\ \sum\limits_{i_1=\max\{i_2+1,start_1\}}^{stop_1} \sum\limits_{i_2=\max\{i_3+1,start_2\}}^{\min\{i_1-1,stop_2\}} \cdots \sum\limits_{i_{fp.r}=start_{fp.r}}^{\min\left\{i_{fp.r-1}-1,stop_{fp.r}\right\}} 1 & \text{else} \end{cases}, \tag{5.21}$$

where

$$v_k = \min\left\{n \in \mathbb{N} : nc_1\pi_1/n_1 \geq kc_0\pi_0/n_0\right\}, \tag{5.22}$$
$$start_k = v_k + k, \tag{5.23}$$
$$w_k = \min\left\{n \in \mathbb{N} : (n + tp.l)c_1\pi_1/n_1 \geq c_0\pi_0/n_0(fp.l + (fp.r - k + 1))\right\} \tag{5.24}$$
$$stop_k = fp.r + tp.r - (w_k + (fp.r - k)). \tag{5.25}$$

The name of the implementation of these two recursive schemes is `posComp.cpp`.

$$|R^{-,\leftarrow}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\dots,n\} : v_k + fnl - k + 1 > fn.l + tn.l - w_k \\ \sum\limits_{i_1=\max\{i_2+1,start_1\}}^{stop_1} \sum\limits_{i_2=\max\{i_3+1,start_2\}}^{\min\{i_1-1,stop_2\}} \cdots \sum\limits_{i_{fn.l}=start_{fn.l}}^{\min\left\{i_{fn.l-1}-1,stop_{fn.l}\right\}} 1 & \text{else} \end{cases}, \tag{5.26}$$

where

$$v_k = \min\left\{n \in \mathbb{N} : nc_0\pi_0/n_0 \geq kc_1\pi_1/n_1\right\}, \tag{5.27}$$
$$start_k = v_k + k, \tag{5.28}$$
$$w_k = \min\left\{n \in \mathbb{N} : (n + tn.r)c_0\pi_0/n_0 \geq c_1\pi_1/n_1(fn.r + (fn.l - k + 1))\right\} \tag{5.29}$$
$$stop_k = fn.l + tn.l - (w_k + (fn.l - k)). \tag{5.30}$$

$$|R^{-,\rightarrow}_{fn,fp}| = \begin{cases} 0 & \text{if } \exists k \in \{1,\dots,n\} : v_k + fn.r - k + 1 \geq fn.r + tn.r - w_k \\ \sum\limits_{i_1=\max\{i_2+1,start_1\}}^{stop_1} \sum\limits_{i_2=\max\{i_3+1,start_2\}}^{\min\{i_1-1,stop_2\}} \cdots \sum\limits_{i_{fn.r}=start_{fn.r}}^{\min\left\{i_{fn.r-1}-1,stop_{fn.r}\right\}} 1 & \text{else} \end{cases}, \tag{5.31}$$

where

$$v_k = \min\left\{n \in \mathbb{N} : nc_0\pi_0/n_0 \geq kc_1\pi_1/n_1\right\}, \tag{5.32}$$
$$start_k = v_k + k, \tag{5.33}$$
$$w_k = \min\left\{n \in \mathbb{N} : (n + tn.l)c_0\pi_0/n_0 \geq c_1\pi_1/n_1(fn.l + (fn.r - k + 1))\right\} \tag{5.34}$$
$$stop_k = fn.r + tn.r - (w_k + (fn.r - k)). \tag{5.35}$$

The name of the implementation of these two recursive schemes is `negComp.cpp`.

5.3.3 Random Permutation

Since the algorithm does not deliver the number of permutations for all points in the support of $\widehat{EIC}$ we will resort to a resampling approach for the estimation of the remaining values of the distribution, see Edgington (2011) for an introduction. This approach establishes the ND by randomly rearranging the class labels N times and calculating the test statistic $\widehat{EIC}((X_i, Y_{p_j(i)})_{i=1,\ldots,n})$ for $j = 1, \ldots, N$, yielding $\widehat{EIC}_1, \ldots, \widehat{EIC}_N$. The probability is then estimated by the share of instances that yields a certain value

$$P\left[\widehat{EIC} = x\right] = \frac{1 + \sum_{j=1}^{N} \mathbb{1}_{(\widehat{EIC}_j = x)}}{N}.$$

However, even for $N \approx 1 \cdot 10^6$ the number of observed permutations which yield a certain value will be small if $P[\widehat{EIC} = x] < 1 \cdot 10^{-5}$ and, thus, the obtained estimation will be inaccurate. Thus, only values that account for more than 0.1% of all permutations will be estimated by this approach.

5.3.4 Interpolation

For those values in the support of $\widehat{EIC}$ which could not be calculated by means of the algorithm described in Section 5.3.2 and which had too little probability mass to allow an accurate estimation by means of random permutations we will resort to interpolation. In Figure 5.5 we can see that this is the case for two regions - to the left and to the right of the mode. For both regions we will calculate the cumulative probability function and interpolate using a method introduced by Hyman (1983). This algorithm constructs monotonicity preserving cubic Hermite interpolants.

For the left region we can use the values obtained through the algorithm to the left and the permutation approach to the right. For the right region we can use the values obtained through the permutation approach and the rightmost value which must equal 1. The results for such an interpolation task can be seen in Figure 5.5, illustrated by the red points.

5.3.5 Implementation

A flow chart of the implementation of the recursive schema can be found in Figure 5.6. The correctness of its implementation can only be assessed for small sample sizes. As for $\widehat{ETC}$ this was achieved by generating the matrix of all possible permutations and applying ψ row-wise. This was undertaken by means of the `allPerm` function of the `multicool` package. The algorithm is described in Williams (2009). The code for these tests can be found in `eic_tests.R`.

The described method has been implemented for the R software platform and is also part of the `UNIC` package. The challenges were the same as for $\widehat{ETC}$. The problem of repeatedly calling the recursive function with the same arguments was tackled with

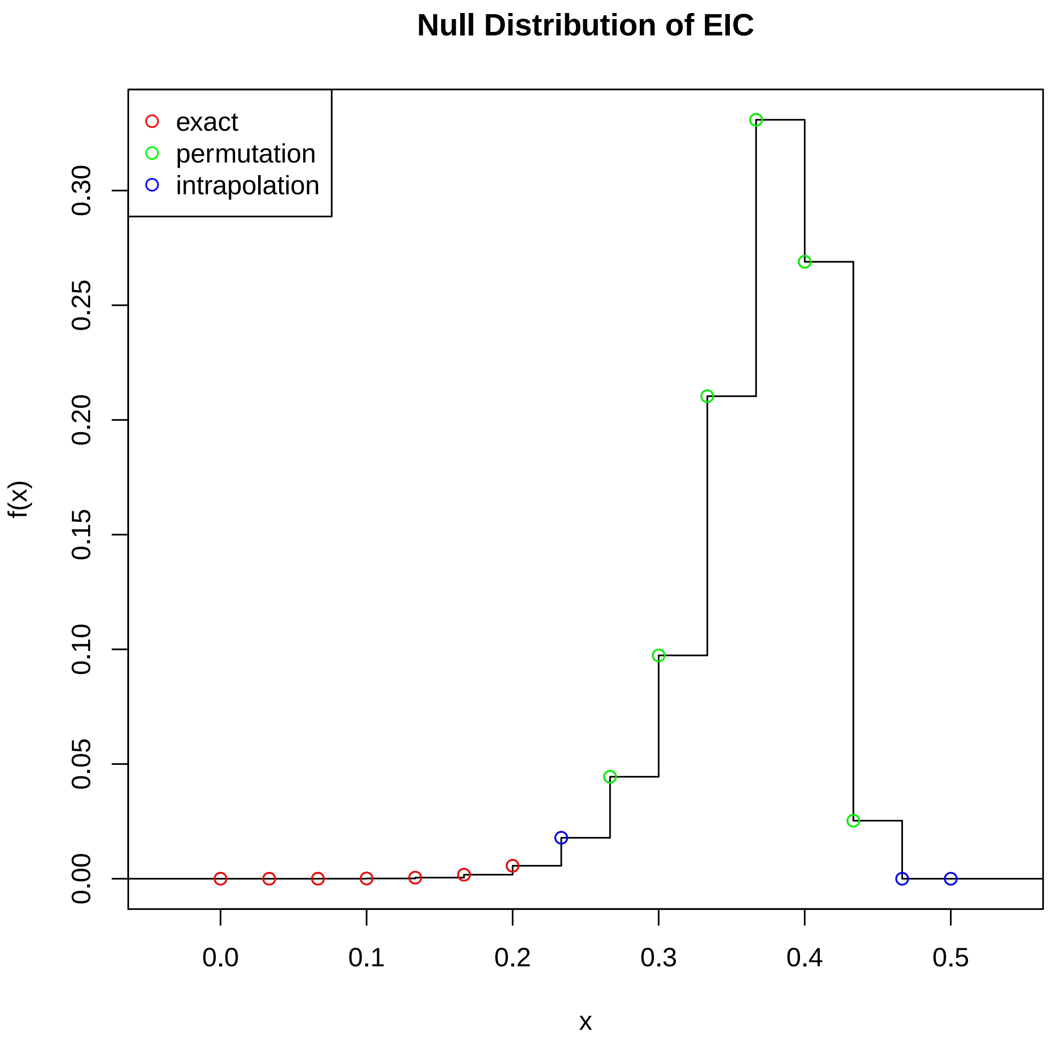


Figure 5.5: This figure illustrates how the three different approaches used to generate the ND, an exact counting algorithm, random permutations, and interpolation are spread over the support. The parameters are $n_0 = 15$, $n_1 = 15$, $c_0 = 1$, $c_1 = 3$, and $\pi_0 = 0.5$.

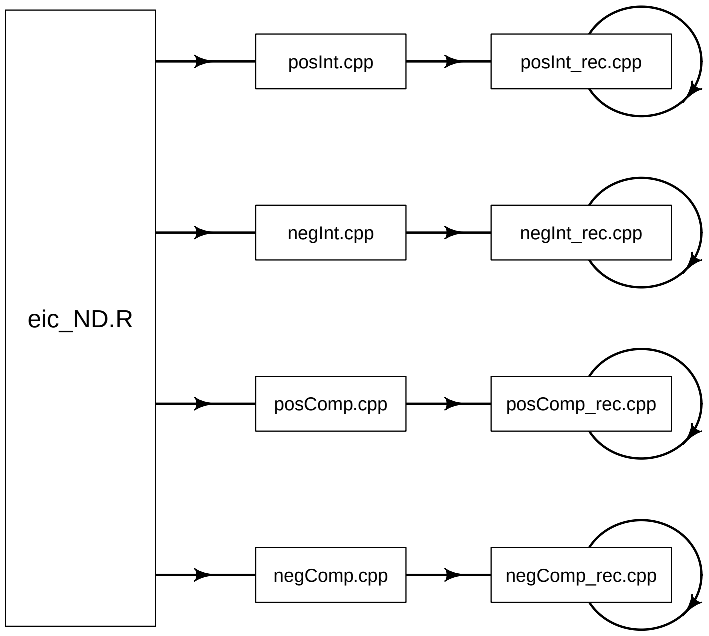


Figure 5.6: This figure illustrates the structure of functions used for the calculation of the null distribution of $\widehat{EIC}$. Different functions are required for the calculation of the favorable permutation of the positive interval and complement as well as the negative interval and complement. These functions further call recursive functions, calling themselves until the base case is reached.

memoization. The problem that the space allocated for integers is not sufficient for the number of permutations. This happens when the sample contains more than 30 instances. Thus, high precision numbers are needed to run the algorithm and save its results. The GNU multiple precision arithmetic library GMP (Granlund et al., 1993–2017) was used.

CHAPTER 6

Simulation Studies

This section intends to shed light on the capacity of EBC_T, EBC, ETC, EIC, and other comparable filters to select signal variables out of a large number of noise variables. Filters are generally used in two ways. Either they are used as a ranking statistic and the number of selected variables is based on other criteria, or the p-value of the statistic is used to further decide on the number of selected variables. We will study both of these capacities.

The first capacity will be quantified by the percentage of signal variables among the top ranking variables, which we will call the *filtering performance* (FP). More formally, let us generate 1000 signal variables and 99000 noise variables and draw a sample of size n from each

$$\begin{array}{lcll} X_{ij} & \sim & F^1 & i = 1, \dots, n_1; j = 1, \dots, 1000 \\ X_{ij} & \sim & F^0 & i = n_1 + 1, \dots, n; j = 1, \dots, 1000 \\ X_{ij} & \sim & F & i = 1, \dots, n; j = 1001, \dots, 100000 \end{array} . \tag{6.1}$$

Thus, a signal variable is one that exhibits differing CCDs ($F^0 \neq F^1$), whereas for a noise variable the CCDs are identical. Since the true underlying distributions are known, $Y_{ij} = 1$ for $j \in 1, \dots, n_1$ and $Y_{ij} = 0$ for $j \in n_1 + 1, \dots, n$. The performance of a filter f for setting (6.1) can then be defined as

$$\text{Performance}(f) := \frac{1}{1000} \sum_{k=1}^{1000} \mathbb{1}(R_k < 1000),$$

where R_k is the rank of the k-th variable, if the filter f is monotone in its arguments and yields small values to good random variables.

The second capacity will be quantified by the percentage of signal variables that are labeled as insignificant by the filter, also referred to as false negative variables. Since the number of false positives is highly influenced by the number of noise variables in the data set, we would like to blend out this effect. Thus, we will simulate 1000 signal variables and observe the percentage of variables that exhibit a p-value above 0.05.

$$\% \text{ False Negatives} = \frac{1}{1000} \sum_{k=1}^{1000} \mathbb{1}(p_k > 0.05),$$

where p_k denotes the p-value of the k-th variable. Since the results of the two approaches are often comparable we will not plot the results for all settings but only for a selected few.

The methods under scrutiny are EBC_T, the prediction error of the LDA, EBC, the prediction error of the QDA, ETC and EIC. Performance will serve as a proxy for different qualities of the filters. The power will be considered in simulation studies A and B, robustness to outliers in simulation study C and skewness in simulation study D. The final experiment of this Chapter (simulation study E) will answer the question how the performance of the filter statistics influences the classifier based on the filtered subset of variables.

6.1 Simulation Study A : Power

In simulation study A, we will study the power of the methods under scrutiny. Since the performance and the power of the statistics are closely related we can observe how the performance of the filters increases with increasing sample size. In this experiment, characterized in (6.2), the signal variables with a sample size of $n \in [10, 200]$, where $n_1 = n_0 = n/2$, are drawn from two Gaussian distributions $N(\mu_1, 1)$ and $N(\mu_0, 1)$, which differ only with respect to their central location, $\Delta\mu = \mu_1 - \mu_0 \in [0, 2.5]$.

$$\begin{array}{lcll} X_{ij} & \sim & \mathcal{N}(\mu_1, 1) & i = 1, \ldots, n_1; j = 1, \ldots, 1000 \\ X_{ij} & \sim & \mathcal{N}(\mu_0, 1) & i = n_1 + 1, \ldots, n; j = 1, \ldots, 1000 \\ X_{ij} & \sim & \mathcal{N}(0, 1) & i = 1, \ldots, n; j = 1001, \ldots, 100000 \end{array} \tag{6.2}$$

Under these assumptions, the LDA is the Bayes classifier and, thus, EBC_T will serve as the benchmark to evaluate the loss in filtering performance of ETC and EBC, since it represents the smallest obtainable value. The results can be found in Figures 6.10 and 6.11 in the appendix. To illustrate the differences more clearly, Figure 6.1 depicts the performance differences of EBC, ETC and EIC to the benchmark EBC_T. The left column in Figure 6.1 assumes $c_1 = 1$ and the second $c_1 = 6$. The rows illustrate the performance differences of EBC, ETC, and EIC, respectively. Not, surprisingly, the nonparametric methods have a smaller power and, thus, their performance is inferior when the sample size is limited. As the sample size increases this effect initially gets larger. After a certain sample size depending on $\Delta\mu$ the difference weakens and diminishes completely. For a better illustration of the effects, however, these sample sizes were excluded from the figure.

EBC shows a slightly inferior power which stems from the fact that is has to estimate two variances with only half the sample size than EBC_T. However, by comparing the second column to the first, it also becomes clear that with an increasing cost inequality the performance difference of EBC increases faster than the nonparametric methods. This seems to indicate that as the costs become more unequal small errors in the estimation of the variances affect the estimation of the prediction error more heavily.

6.2 Simulation Study B : Power

In simulation study B, characterized in (6.3), we will generalize simulation setting A by relaxing the assumption of equal variances of the Gaussian class conditionals. We, thus, add one more dimension to the simulation experiment $\log_2(\sigma_1/\sigma_0)$, where $\sigma_1 \in 2^{[-3,3]}$ and $\sigma_0 = 1/\sigma_1$. Since this makes a visual inspection impossible, we will hold the sample size fixed at a level ($n = 100$) where the performance differences between the filters manifest. As in simulation study A we will allow the difference in the central location to differ, $\Delta\mu = \mu_1 - \mu_0 \in [0, 2.5]$.

$$\begin{array}{lll} X_{ij} \sim & \mathcal{N}(\mu_1, \sigma_1^2) & i = 1, \ldots, 50; j = 1, \ldots, 1000 \\ X_{ij} \sim & \mathcal{N}(\mu_0, \sigma_0^2) & i = 51, \ldots, 100; j = 1, \ldots, 1000 \\ X_{ij} \sim & \mathcal{N}(0, 1) & i = 1, \ldots, 100; j = 1001, \ldots, 100000 \end{array} \tag{6.3}$$

The results of simulation study B for varying costs c_1 can be found in Figures 6.2 and 6.3 ($c_1 = 1$) and Figures 6.12, 6.13, 6.14, and 6.15 ($c_1 = 3, 6$) in the appendix. These results clearly indicate that there are systematic differences in the performance of the filter statistics when the assumption of equal variances is not met. Intuitively, the stronger the CCDs differ from another, the easier it should be to predict the class based on the observation of X. This characteristic can be observed for EBC, which yields the prediction error the Bayes classifier under the current assumptions, except for the hyperplane characterized by $\sigma_1 = \sigma_0$. The greater the difference between σ_1 and σ_0 the better the filtering performance of EBC, see the top right illustration in Figures 6.2, 6.12, and 6.14. EIC the nonparametric equivalent shows the same trend as EBC with a slightly inferior performance.

As we move away from the hyperplane $\sigma_1 = \sigma_0$, the Bayes classifier is not a member of the family of threshold classifiers, since the optimal positive and negative domains are an interval and its respective complement. The two filters based on threshold classification, thus, become biased. However, this bias manifests differently. The performance of EBC_T deteriorates equally when we move away from the hyperplane and this trend is independent of the costs. Contrary to EBC_T, the performance of ETC depends strongly on the misclassification costs. While for $c_1 = c_0$ ETC shows no systematic bias, for $c_1 = 3, 6$ the filter performance deteriorates completely on one side of the hyperplane. ETC can benefit from differences in the variances of the CCDs if the positive class exhibits the relatively lower variance and $c_1 > c_0$. This becomes clear, if one considers the extreme case of $\sigma_1 = 0$, where the positive CCD degenerates to a single point. Even

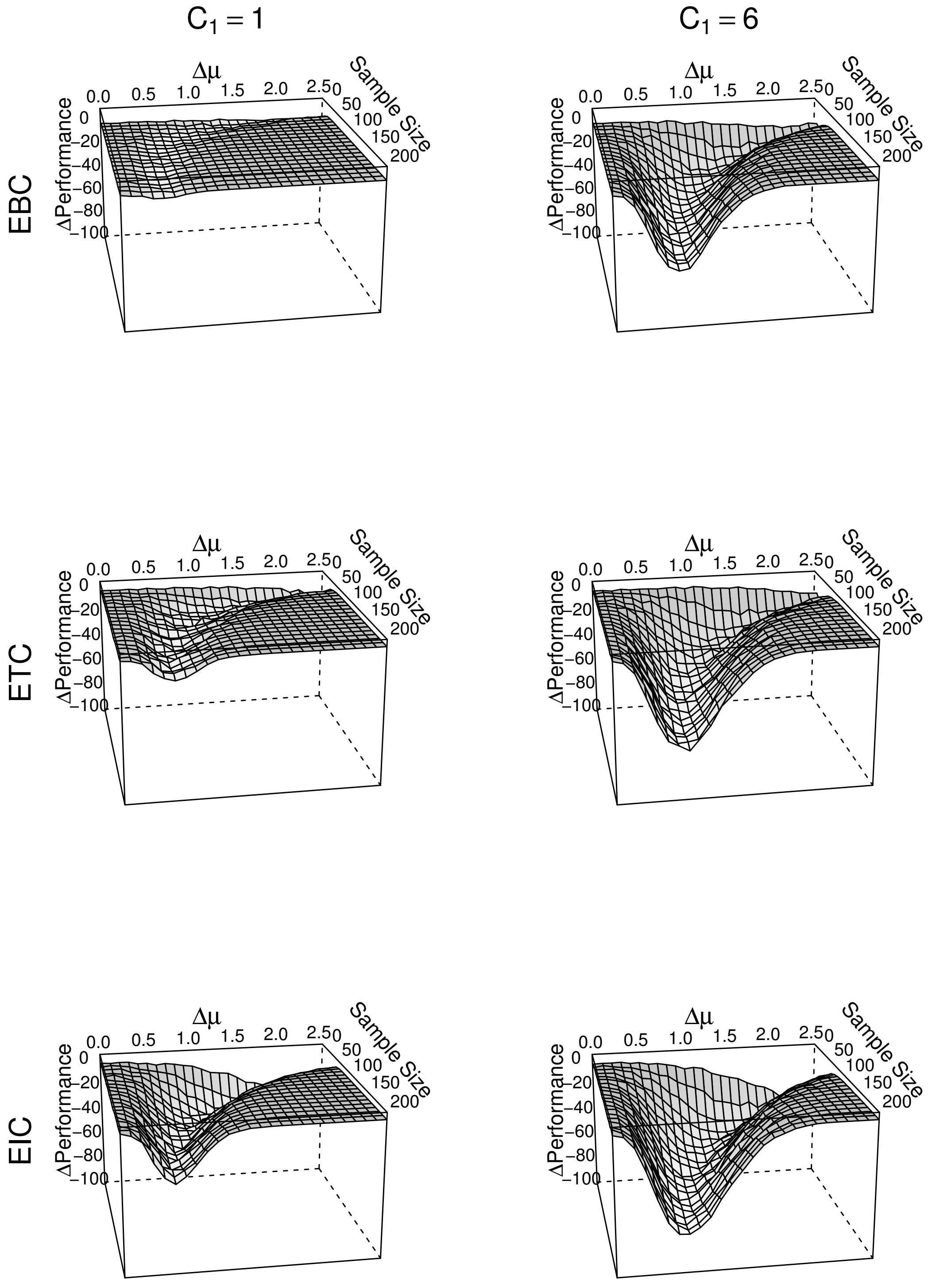


Figure 6.1: Simulation Study A studies the differences in power of the filter statistics under scrutiny by means of the performance differences of the filters EBC, ETC, and EIC with respect to EBC_T. This filter yields the prediction error of the Bayes classifier under these assumptions and is, thus, used as a benchmark value. The simulation experiment has been conducted for two different misclassification costs $c_1 = 1$ (left column) and $c_1 = 6$ (right column).

if $\Delta\mu = 0$, one can obtain a false negative rate of 0 and a false positive rate of 0.5 by setting a threshold just below this point.

Further, as the costs become more unequal the region where EBC_T outperforms EBC increases. This is the case when the bias introduced by incorrectly assuming $\sigma_1 = \sigma_0$ is made up by the reduced variance. EBC is then strongly affected, to an extent that it is even outperformed by EIC.

6.3 Simulation Study C : Robustness to Outliers

This experiment, characterized in (6.4), considers data that are contaminated with a varying proportion of outliers, $\phi \in [0, 0.2]$. Furthermore, we shall vary the difference in the central location, $\Delta\mu = \mu_1 - \mu_0 \in [0, 2.5]$. Again the number of samples are fixed at $n = 100$, with an equal number of positives and negatives. In order to blend out the differences in the filtering performance caused by other effects we shall compare the FP with the FP of the data without outliers. Thus, let us define $\Delta\text{FP}(\phi) :=\text{FP}(\phi)-\text{FP}(0)$, where $\text{FP}(\phi)$ denotes the filtering performance with a proportion of ϕ outlying values, ceteris paribus. The results for varying misclassification costs ($c_1 = 3, 1, 6$) are depicted in Figures 6.4 and 6.5 as well as Figures 6.16, and 6.17 in the appendix.

$$\begin{array}{lcll}
X_{ij} & \sim & \mathcal{N}(\mu_1, 1) & i = 1, \ldots, 50 - \lfloor \phi \cdot 50 \rfloor ; j = 1, \ldots, 1000 \\
X_{ij} & \sim & \mathcal{N}(\mu_1, 5) & i = \lfloor \phi \cdot 50 \rfloor + 1, \ldots, 50; j = 1, \ldots, 1000 \\
X_{ij} & \sim & \mathcal{N}(\mu_0, 1) & i = 51, \ldots, 100 - \lfloor \phi \cdot 50 \rfloor ; j = 1, \ldots, 1000 \\
X_{ij} & \sim & \mathcal{N}(\mu_0, 5) & i = 100 - \lfloor \phi \cdot 50 \rfloor + 1, \ldots, 100; j = 1, \ldots, 1000 \\
X_{ij} & \sim & \mathcal{N}(0, 1) & i = 1, \ldots, 100 - \lfloor \phi \cdot 100 \rfloor ; j = 1001, \ldots, 100000 \\
X_{ij} & \sim & \mathcal{N}(0, 5) & i = 100 - \lfloor \phi \cdot 100 \rfloor + 1, \ldots, 100; j = 1001, \ldots, 100000
\end{array} \tag{6.4}$$

Simulation Setting C sheds light on the robustness of the classifiers in the presence of outliers. The performance, defined in 6 will serve as a proxy for robustness, since a higher robustness will consequently improve the performance of a filter. Not surprisingly, the filtering performance deteriorates in all methods under scrutiny when outliers are introduced, however ETC and EIC prove to be fairly robust. This is not surprising, since these methods are based on the ranks and, thus, exhibit a breakdown point close to 0.5. The performance of the default implementations of the parametric filters EBC_T and EBC deteriorate quickly when the data contain outliers. However, the `EBC` function allows substituting the maximum likelihood estimates of the parameters of Gaussian class conditionals by robust estimates. More precisely, if the argument `robust` is set to `TRUE`, the median and MAD are calculated. The robust versions are indicated by $rEBC_T$ and $rEBC$, respectively. One can clearly see that the robust versions even outperform ETC and EIC, especially when the misclassification costs become more unequal.

Surprisingly, ETC is more affected by unequal misclassification costs than other methods. $rEBC_T$ proved to be the most robust filter statistic.

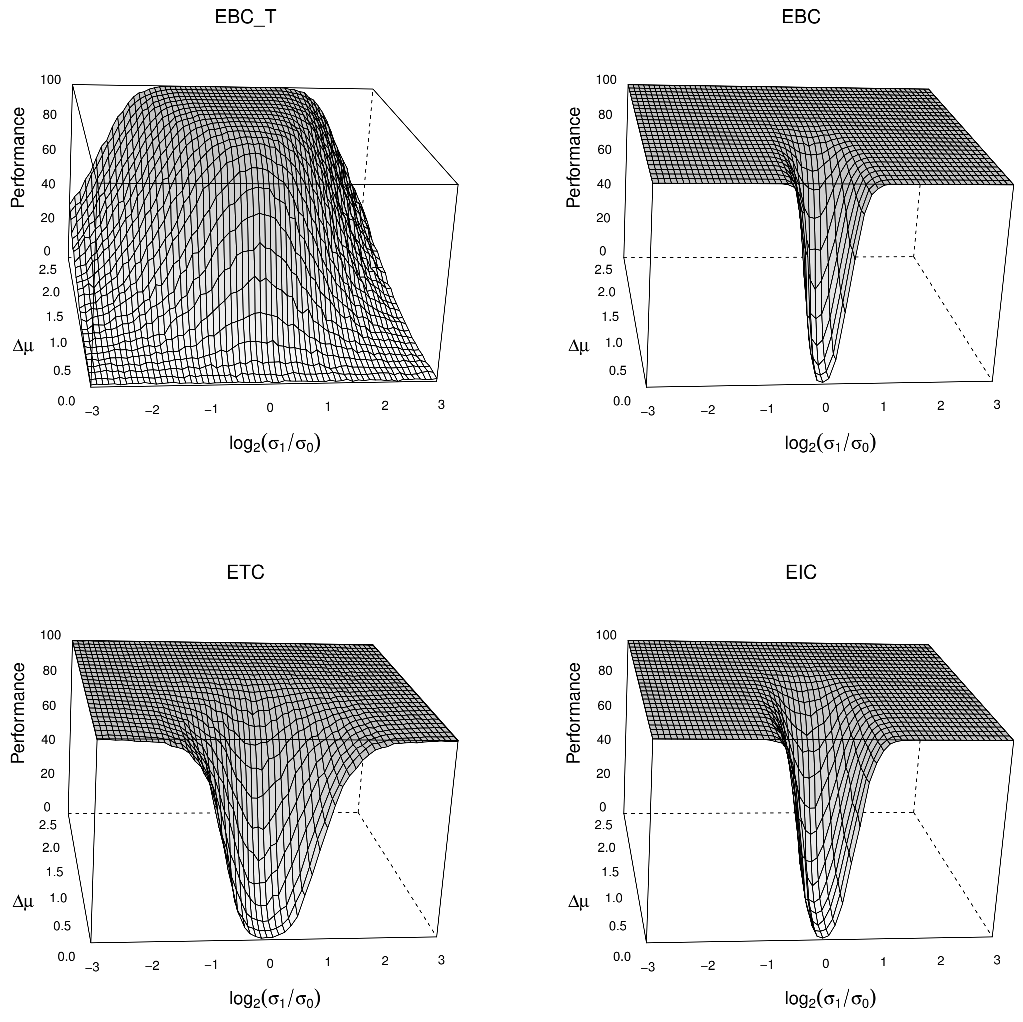


Figure 6.2: This figure illustrates the performance of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 1$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$.

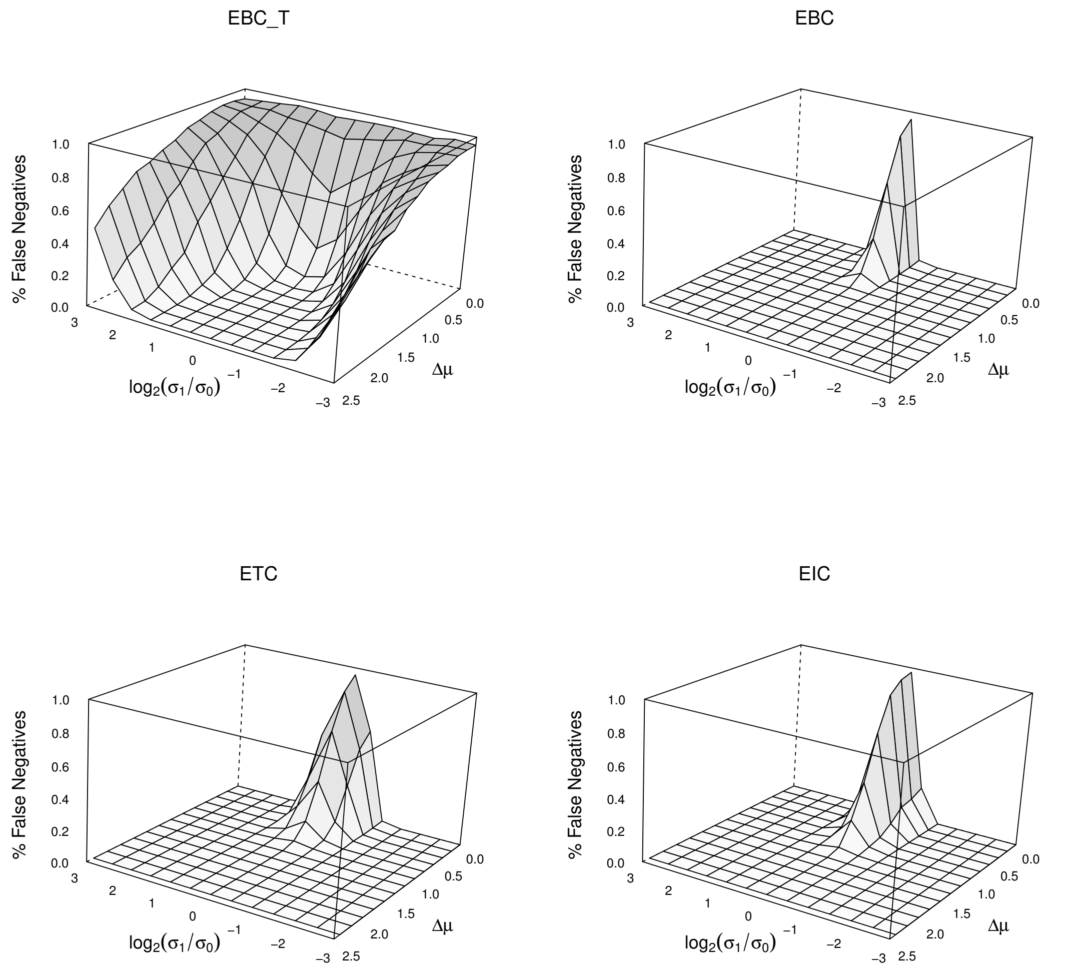


Figure 6.3: This figure illustrates the percentage of false negative variables of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 1$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$. The interpretation of the results is equivalent to the interpretation for the performance.

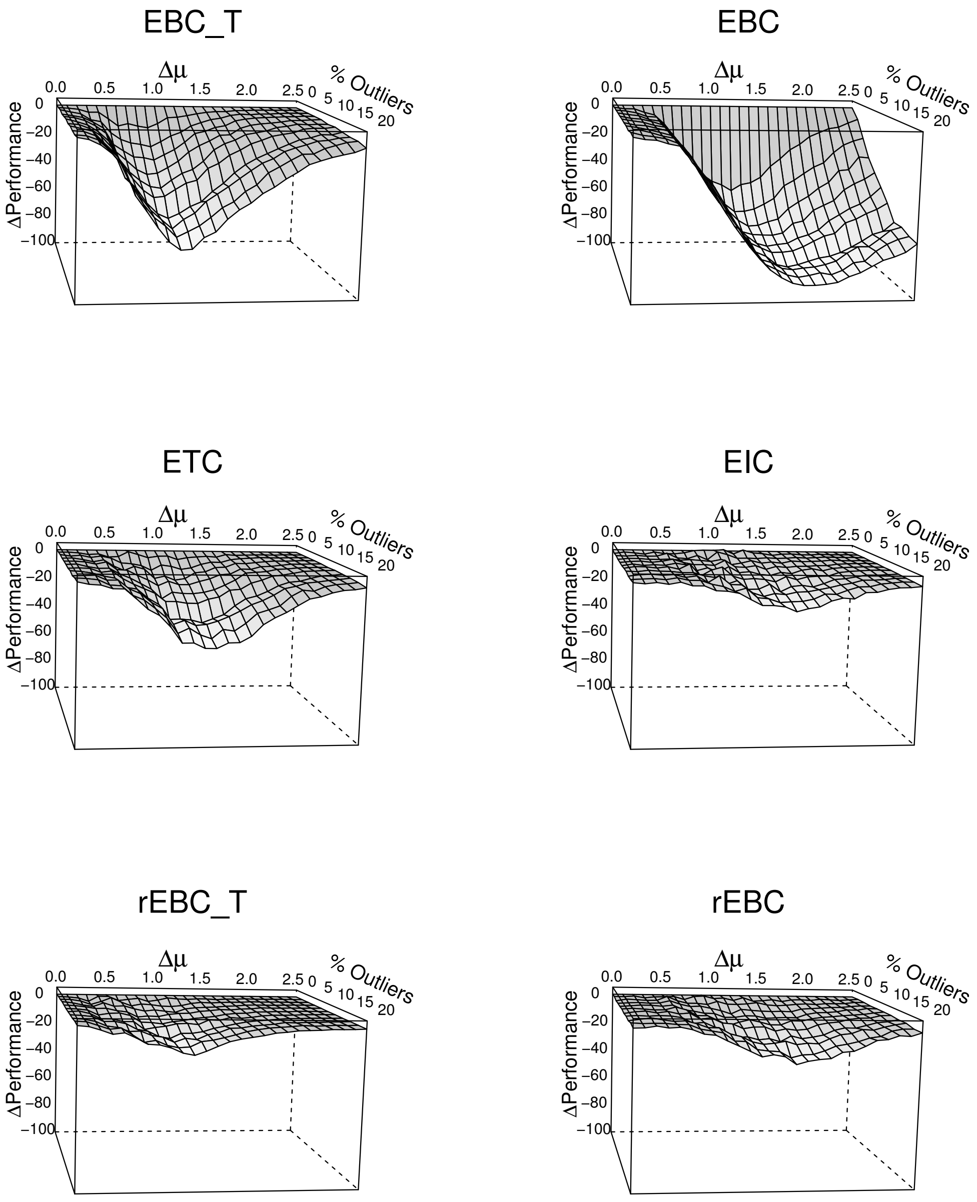


Figure 6.4: Results of Simulation study C for $c_1 = 3$. This experiment analyses the performance of the filters under scrutiny for two Gaussian class conditional distributions that are contaminated with a varying degree of outliers. ΔFP denotes the difference in performance as compared to the simulation study without outliers, all other parameters being equal.

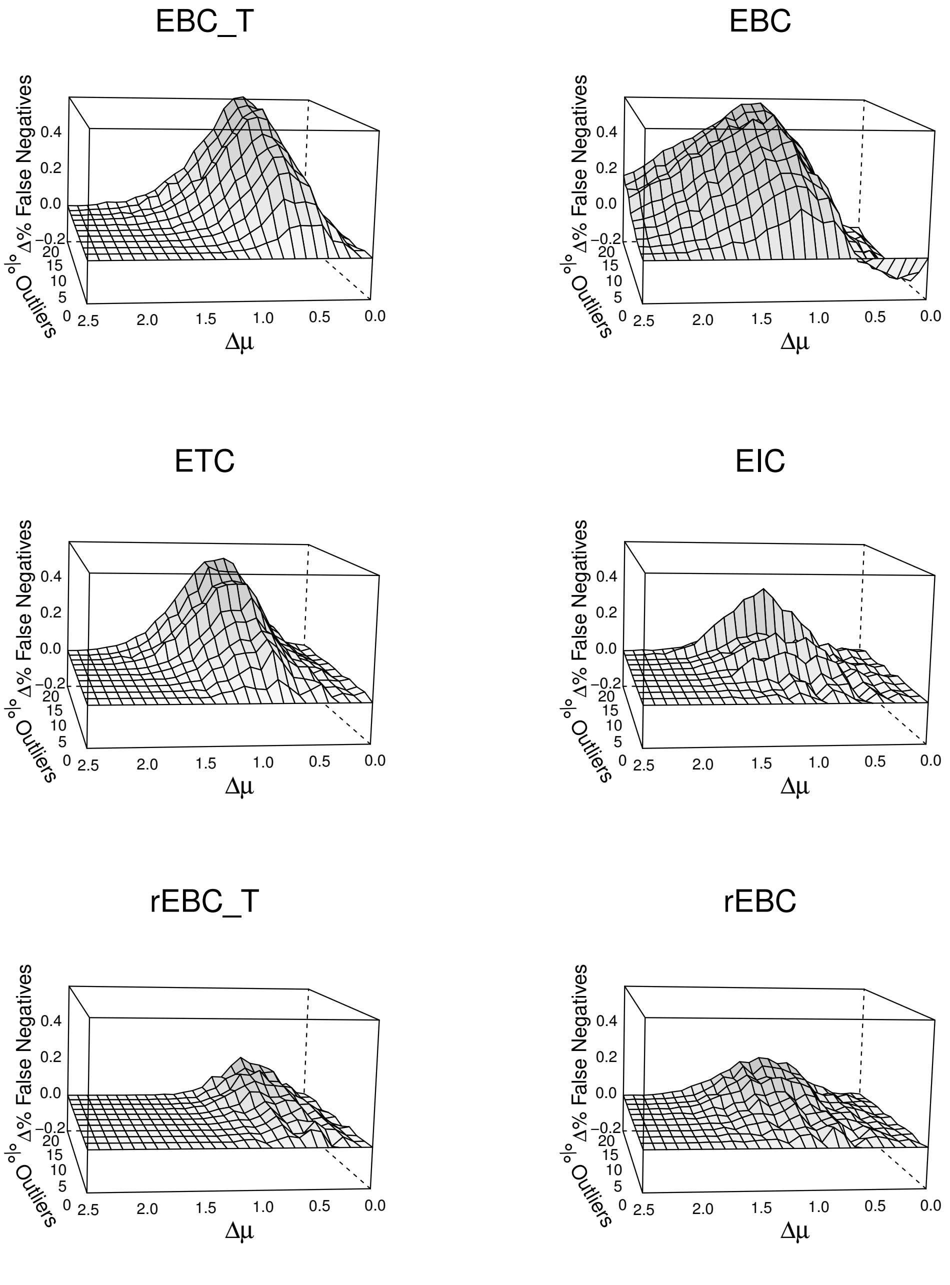


Figure 6.5: Results of Simulation study C for $c_1 = 3$. This experiment analyses the percentage of false negative variables of the filters under scrutiny for two Gaussian class conditional distributions that are contaminated with a varying degree of outliers. Δ % false negatives denotes the difference in percentage of false negatives as compared to the simulation study without outliers, all other parameters being equal. The robustness of rEBC seems to outperform EIC even more.

6.4 Simulation Study D : Robustness to Skewness

In this simulation study, characterized in (6.5), we would like to study the effect of skewness on the filtering performance. The random variables are, thus, drawn from a log-normal distribution with the parameters $\mu = 0$ and $\sigma \in \sqrt{2^{[-3,3]}}$. In order to introduce a difference in the central location, the parameter $\Delta \in [0, 2.5]$ controls the shift in the central location of the distribution. Again, the number of samples is fixed at $n = 100$, with $n_1 = n_0 = n/2$. The results are depicted in Figures 6.6, 6.18 and 6.19 for varying misclassification costs ($c_1 = 1, 3, 6$).

$$\begin{array}{rcll} \ln(X_{ij}) & \sim & \mathcal{N}(0,\sigma^2) & i = 1,\ldots,n_1; j = 1,\ldots,1000 \\ \ln(X_{ij}) & \sim & \mathcal{N}(0,\sigma^2) - \Delta & i = n_1+1,\ldots,n; j = 1,\ldots,1000 \\ \ln(X_{ij}) & \sim & \mathcal{N}(0,\sigma^2) & i = 1,\ldots,n; j = 1001,\ldots,100000 \end{array} \tag{6.5}$$

Simulation Setting D sheds light on the filtering performance of the filter statistics under scrutiny when the CCDs are non-symmetric or skewed. *ETC* and *EIC* are not only completely resilient to deviations from symmetry, their performance seems to even improve slightly. EBC_T and *EBC* clearly show a strong deterioration in the filtering performance, which is independent of the misclassification costs.

Comparing the results of Figures 6.6 and 6.7 one clear difference stands out. While the performance of EBC drops to 0 when $\log_2(\sigma) > 0$ the percentage of false negatives does not deteriorate equally dramatically. This seems to suggest that EBC struggles with false positive variables. While the number of false negatives drops to 0 as $\log_2(\sigma)$ goes to three the performance still deteriorates. This can only occur if the number of false positive variables increases and as a result the number of positives among to top ranked variables decreases.

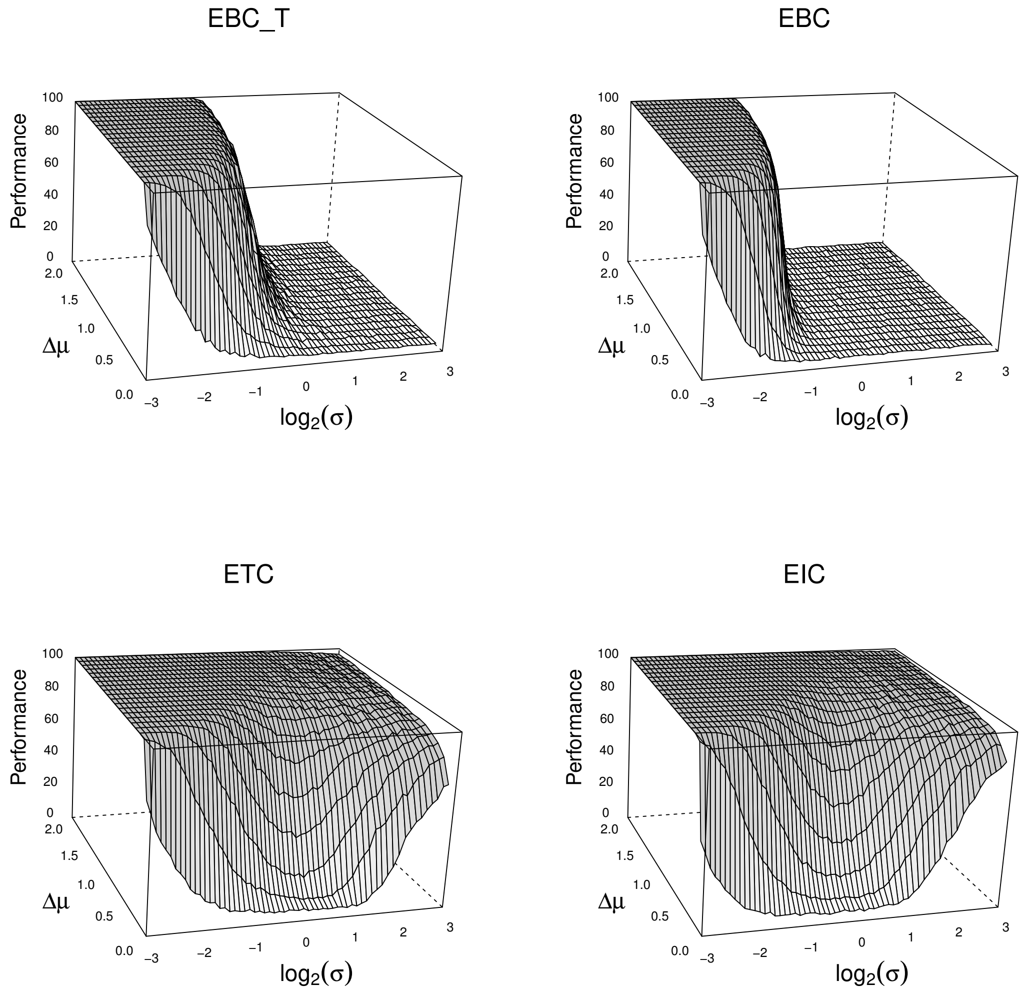


Figure 6.6: Results of Simulation study D for $c_1 = 1$. This experiment analyses the performance of the filter statistics under scrutiny for two lognormal class conditional distributions with varying degrees of skewness.

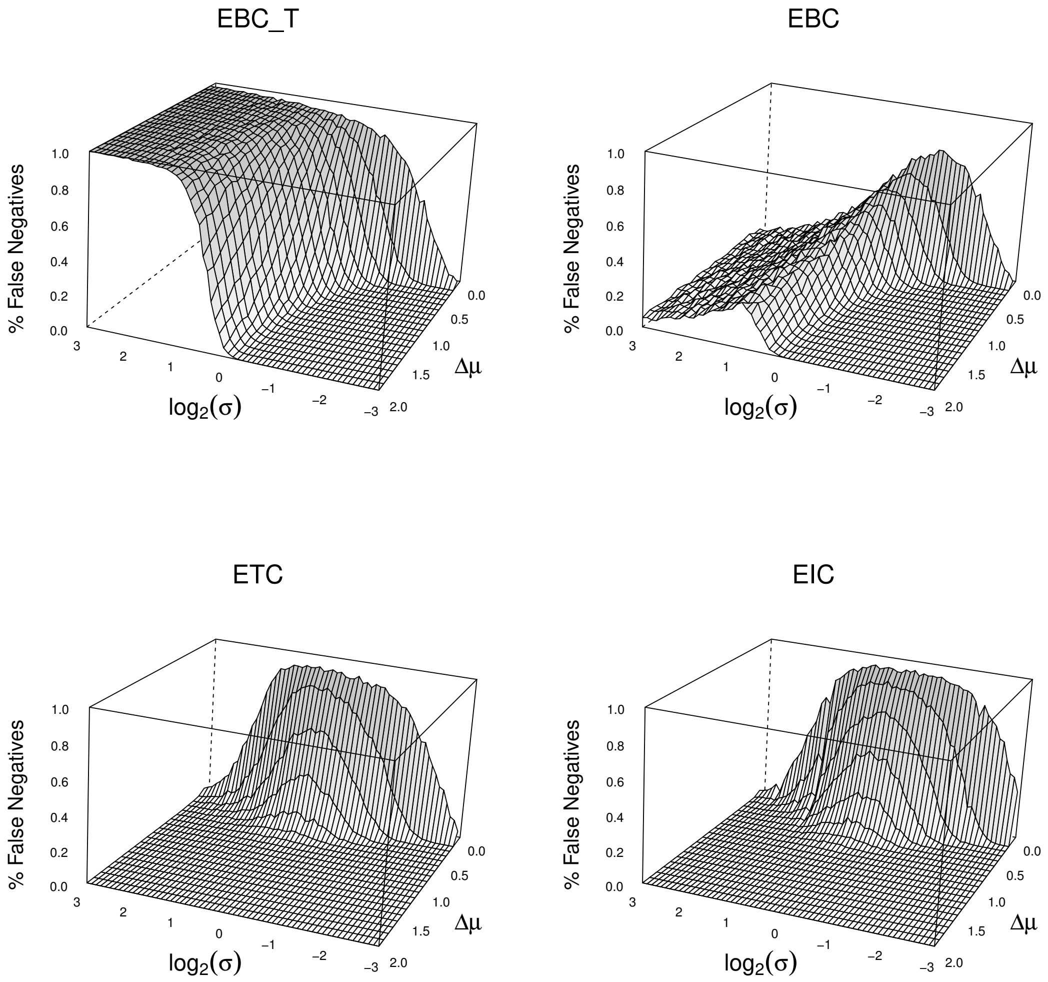


Figure 6.7: Results of Simulation study D for $c_1 = 1$. This experiment analyses the percentage of false negative variables of the filter statistics under scrutiny for two lognormal class conditional distributions with varying degrees of skewness. Note, that in order to better illustrate the function in three dimensions the position of the x and y-axis were inverted compared to Figure 6.6

6.5 Simulation Study E : Model Selection

The performance of a variable filter can not only be evaluated by the share of signal variables it succeeded to identify but also by the performance of a multivariate classifier which builds models using the selected subset of variables. This section is, thus, dedicated to analyzing the predictive power when filtering is used as a model selection method.

The simulation study, characterized in (6.6), simulates 40 signal variables and 960 noise variables. The signal variables are drawn from mixtures of Gaussian class conditionals and fall into five groups depending on the relative variances $\log_2(\sigma_1/\sigma_0) \in \{-2, -1, 0, 1, 2\}$. There are, thus, variables with equal variances and variables with varying degrees of differing variances. The differences in the central locations are $\Delta\mu = \mu_1 - \mu_0 \in \{0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4\}$. One variable is drawn for every combination of $\Delta\mu, \log_2(\sigma_1/\sigma_0)$, yielding 40 signal variables over all. The operating conditions for this simulation study were set to $oc = c(5, 1, 0.5)$

$$\begin{array}{lll} X_{ij} \sim & \mathcal{N}(\mu_1, \sigma_1^2) & i = 1, \dots, 50; j = 1, \dots, 40 \\ X_{ij} \sim & \mathcal{N}(\mu_0, \sigma_0^2) & i = 51, \dots, 100; j = 1, \dots, 40 \\ X_{ij} \sim & \mathcal{N}(0, 1) & i = 1, \dots, 100; j = 41, \dots, 1000 \end{array} \tag{6.6}$$

Three independent data sets with these specifications were generated: one for filtering, one for fitting the multivariate classification model using only the subset of variables, and one for prediction. All five different filters were applied to the exact same datasets and the resulting ranks were recorded. For subsets ranging from one to ten top ranked variables a best subset model selection was implemented. This means, that for every possible combination of variables in the subset a classifier is fitted. Consequently, the fitted model is used to predict the instances of the third data set. The classifier is a multivariate QDA that is OC-sensitive. The results are depicted in Figure 6.8.

From Figure 6.8 we can see that all filters obtain perfect classification results at a model size of ten. The smaller the cardinality of the subset the weaker the predictive power of the model. However, the performance of the filters fall into two groups. A weak performance is shown by the t-test and EBC_T. Those filters will probably select variables with equal variances, which will experience difficulties in building good OC-sensitive classifiers. The second group of filters comprised of EBC, ETC, and EIC. These filters tend to select variables with relative variances suitable for the general classification task and will, thus, build stronger models. Even though, this represents a stylized example, its results can clearly emphasize the importance of OC-sensitive filtering.

Figure 6.9 depicts other interesting aspects of the filters. The ranks of the t-test and EBC_T show perfect correlation for the top 350 variables. This is not surprising since they are both functions of the sample mean and variance. The ranks of the lower ranking variables, however, seem to be completely uncorrelated.

The correlation of the ranks of the t-test with the other methods is very weak (=0.16), see Table 6.1. Among the top ranked variables there is some correlation but the ranks

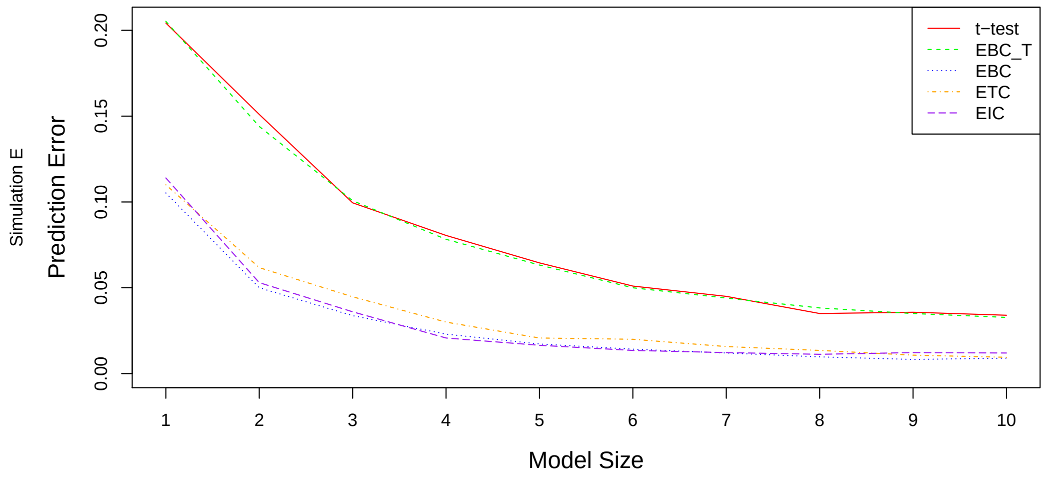


Figure 6.8: illustrates the results of Simulation Study E. The x-axis depicts the model size and the y-axis the expected prediction error. Every curve depicts one filtering method. As the number of variables increases, the prediction error decreases. In this stylized example all filters exhibit a perfect discrimination between the two classes at a model size of 10. However, one can see that the variables selected by EBC, ETC, and EIC show a better performance with fewer variables, which emphasizes the importance of cost-sensitive filtering.

among the lower ranks seem to be completely independent. The same holds true for the correlation of EBC_T with the other filters. The overall correlation is negligible, only among the top ranked variables there is some similarity. Interesting, because unexpected is the correlation between EBC and ETC (=0.62). In Simulation Study B we have learned that ETC can benefit of unequal variances only if the positive class has a smaller variance. Thus, for those variables the correlation of the ranks will be high. For all other variables the correlation is supposedly very low.

The perfect fit between the higher ranked variables of EBC_T and EBC has an easy explanation. If the discriminatory power of a variable is very low, the classifier degenerates and the expected prediction error will be set to $\min\{c_0\pi_0, c_1\pi_1\}$. Thus, all these variables have the same EPE and consequently the rank is given by the position in the vector. To be more precise, if there are a number of variables with the same test statistic, the variable with lowest index number among those will be assigned the lowest rank, the variable with the second lowest index number, the second lowest rank and so on.

The lines in the scatter plots of ETC and EIC are caused by the discrete nature of these filters. There is only a finite number of values the test statistic can take and, thus, there will be a high number of variables sharing the same rank.

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.86	1			
EBC	0.17	0.17	1		
ETC	0.15	0.15	0.62	1	
EIC	0.16	0.18	0.12	0.13	1

Table 6.1: Spearman's rank correlations of the ranks of five different filtering statistics.

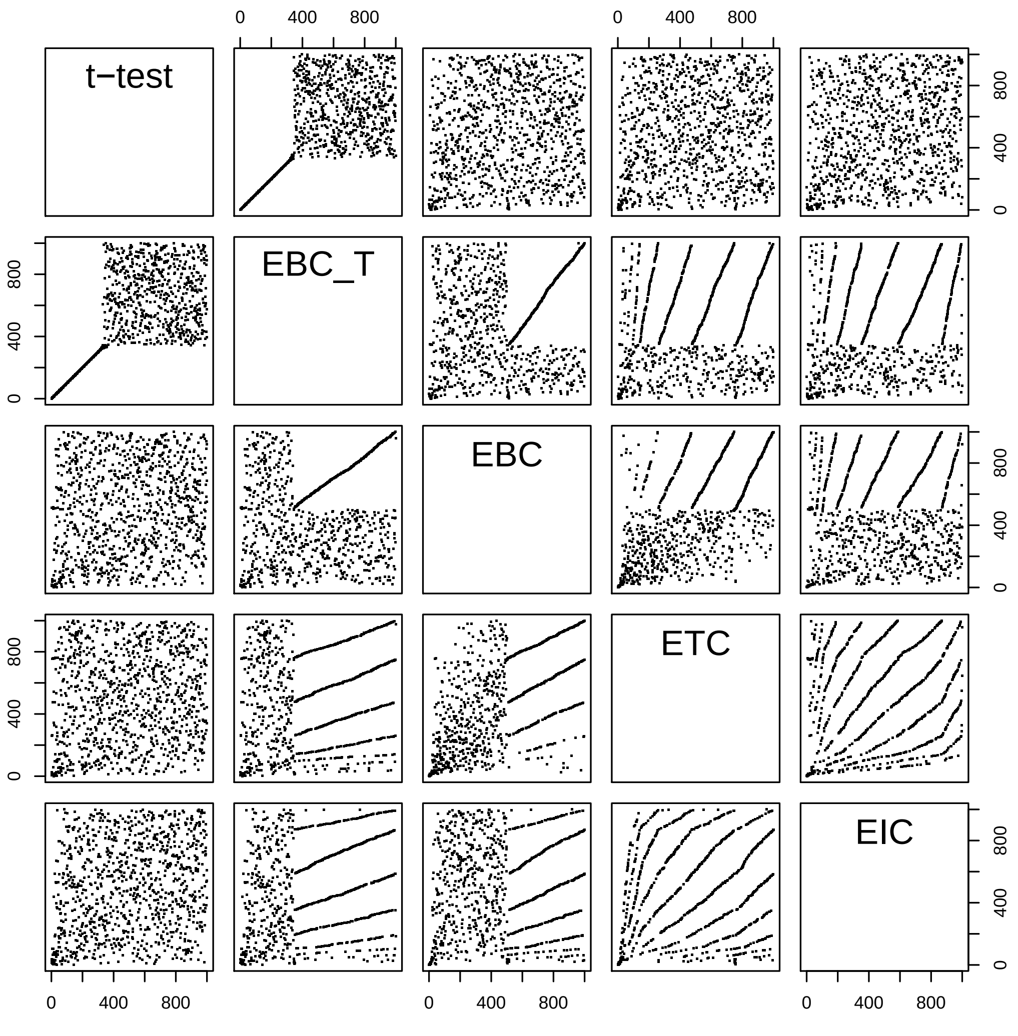


Figure 6.9: plots the ranks of the different filter statistics against each other. The most striking fact is that the ranks of the t-test and EBC_T exhibit a perfect correlation for the 400 top ranking features.

6.6 Supplementary Figures

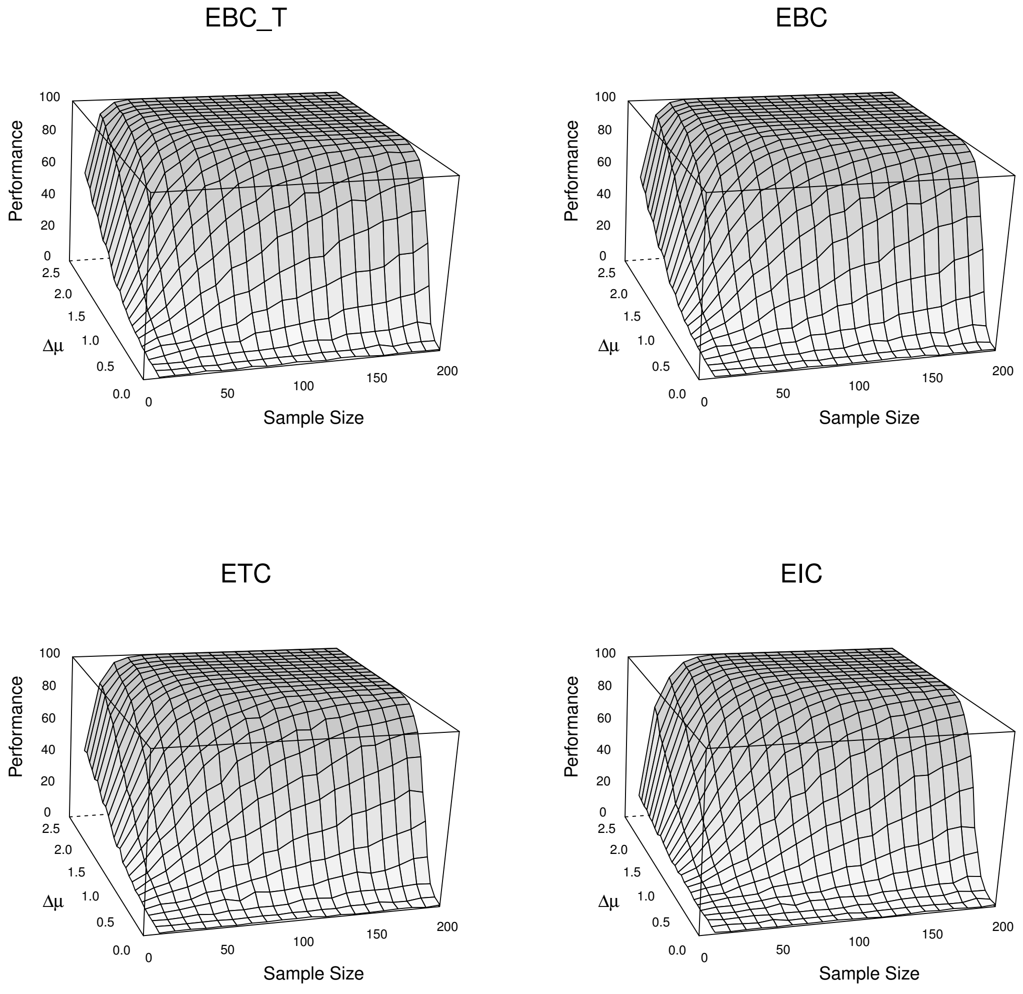


Figure 6.10: This Figure illustrates the performance for all filters under scrutiny for Gaussian class conditionals with equal variances and a varying difference in the central location. EBC_T yields the prediction error of the Bayes classifier under these assumptions and we can see that this filter exhibits the greatest power. Its performance increases the fastest with increasing sample size.

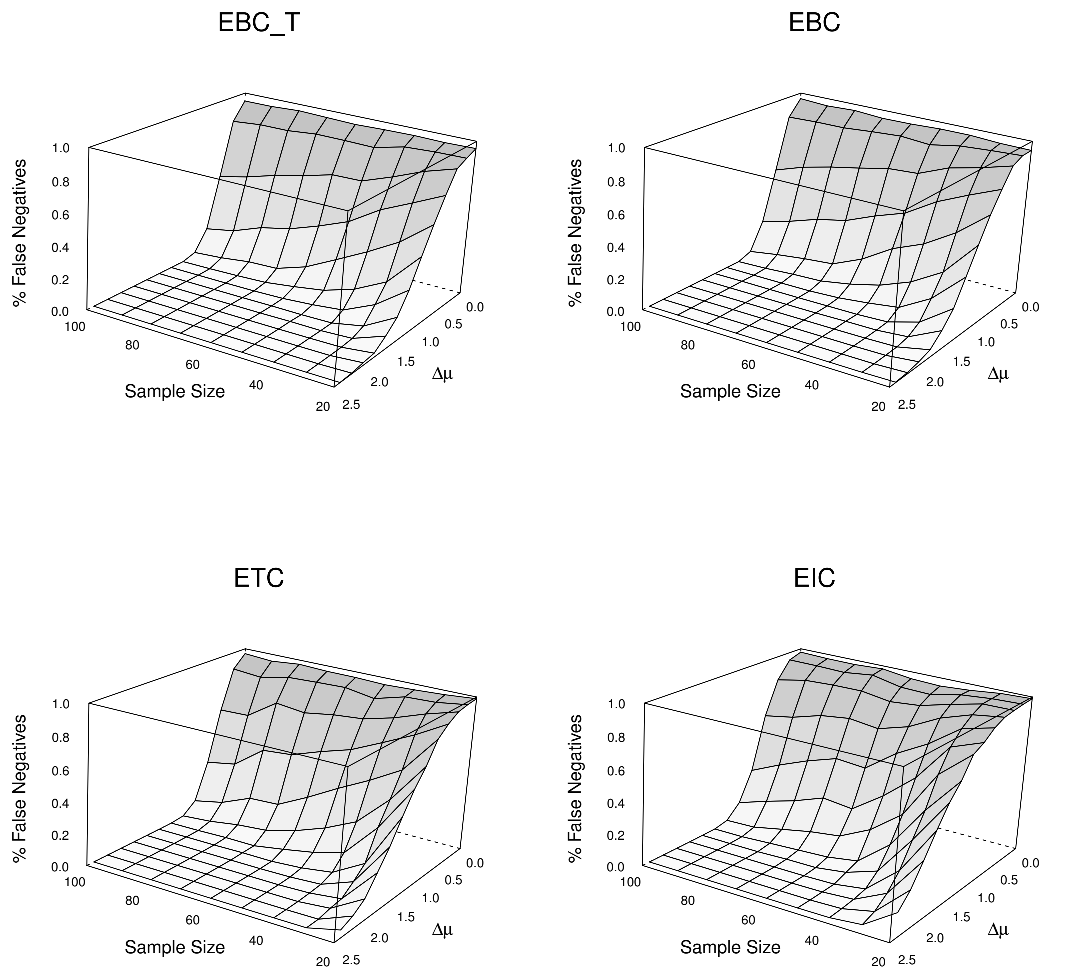


Figure 6.11: This Figure illustrates the percentage of false negatives for all filters under scrutiny for Gaussian class conditionals with equal variances and a varying difference in the central location. EBC_T yields the prediction error of the Bayes classifier under these assumptions and we can see that this filter exhibits lowest share of false negatives.

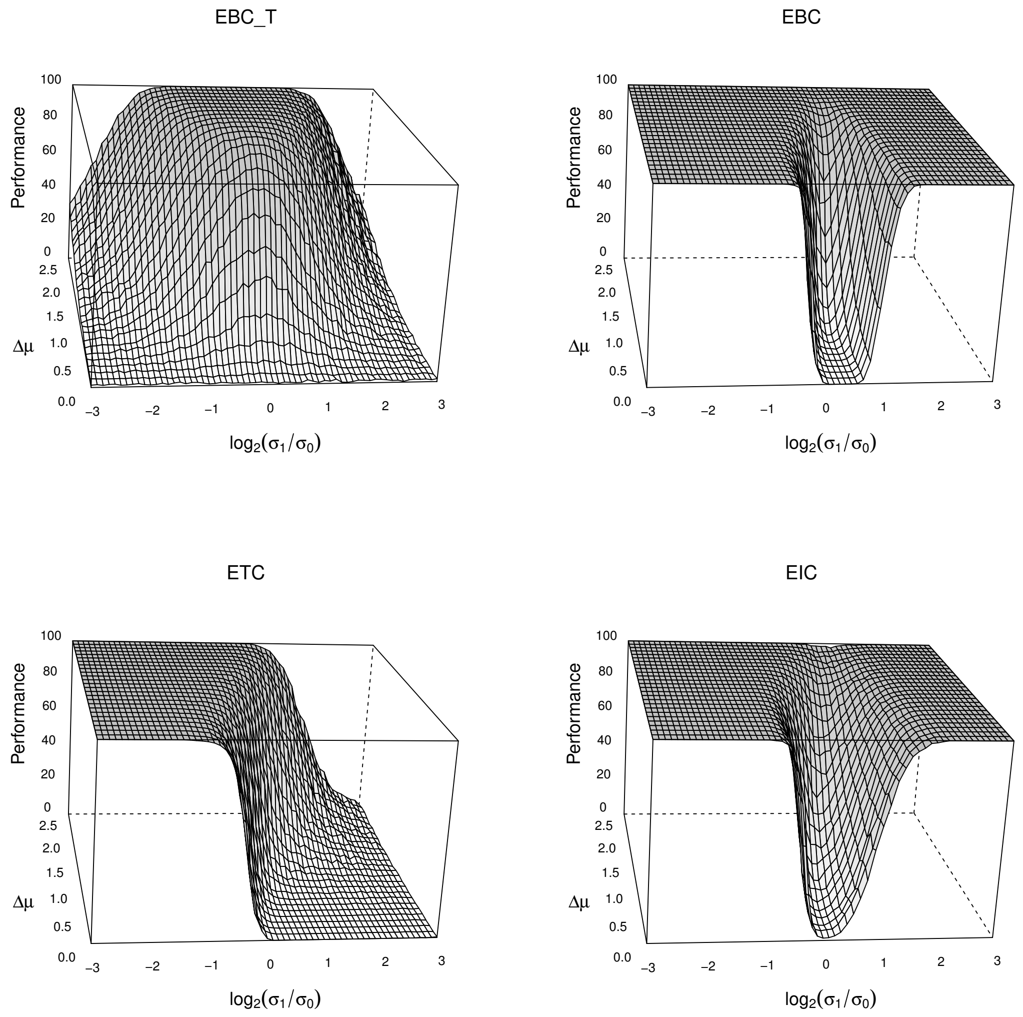


Figure 6.12: This figure illustrates the performance of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 3$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$.

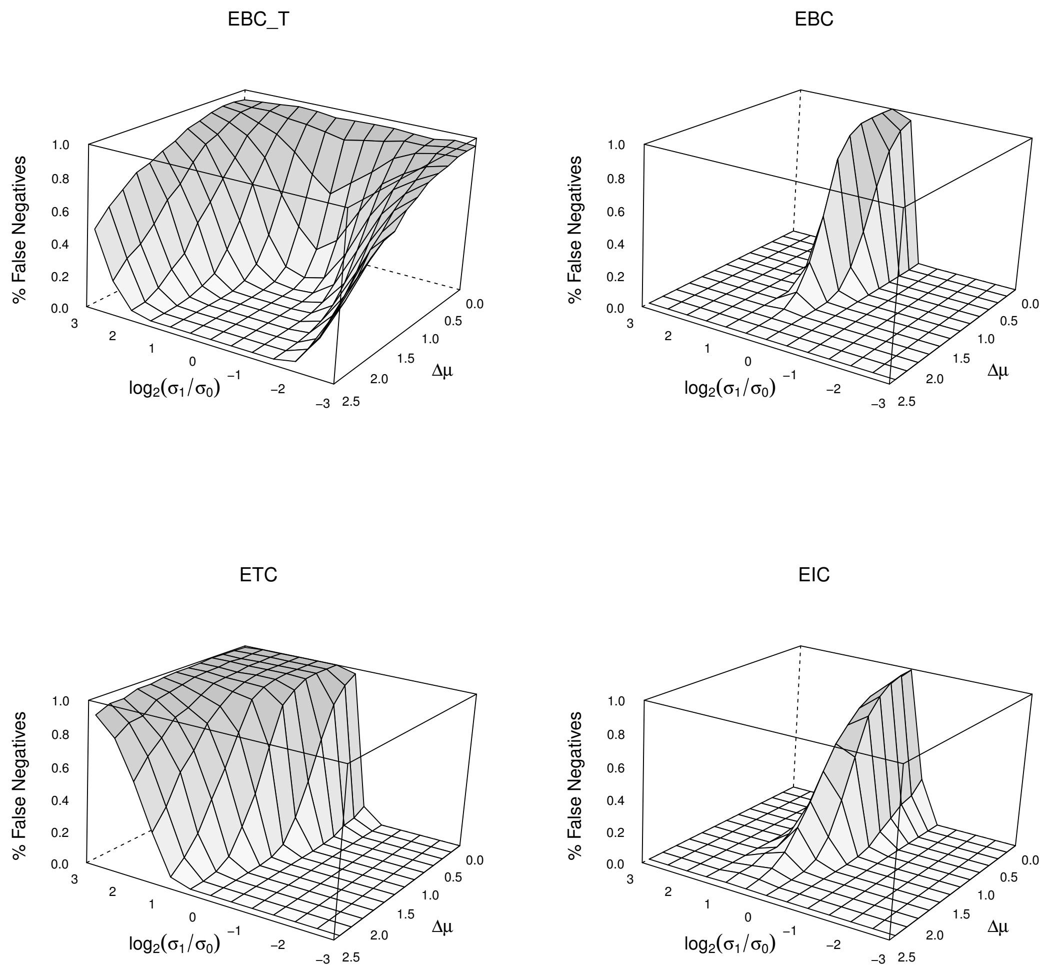


Figure 6.13: This figure illustrates the percentage of false negatives of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 3$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$.

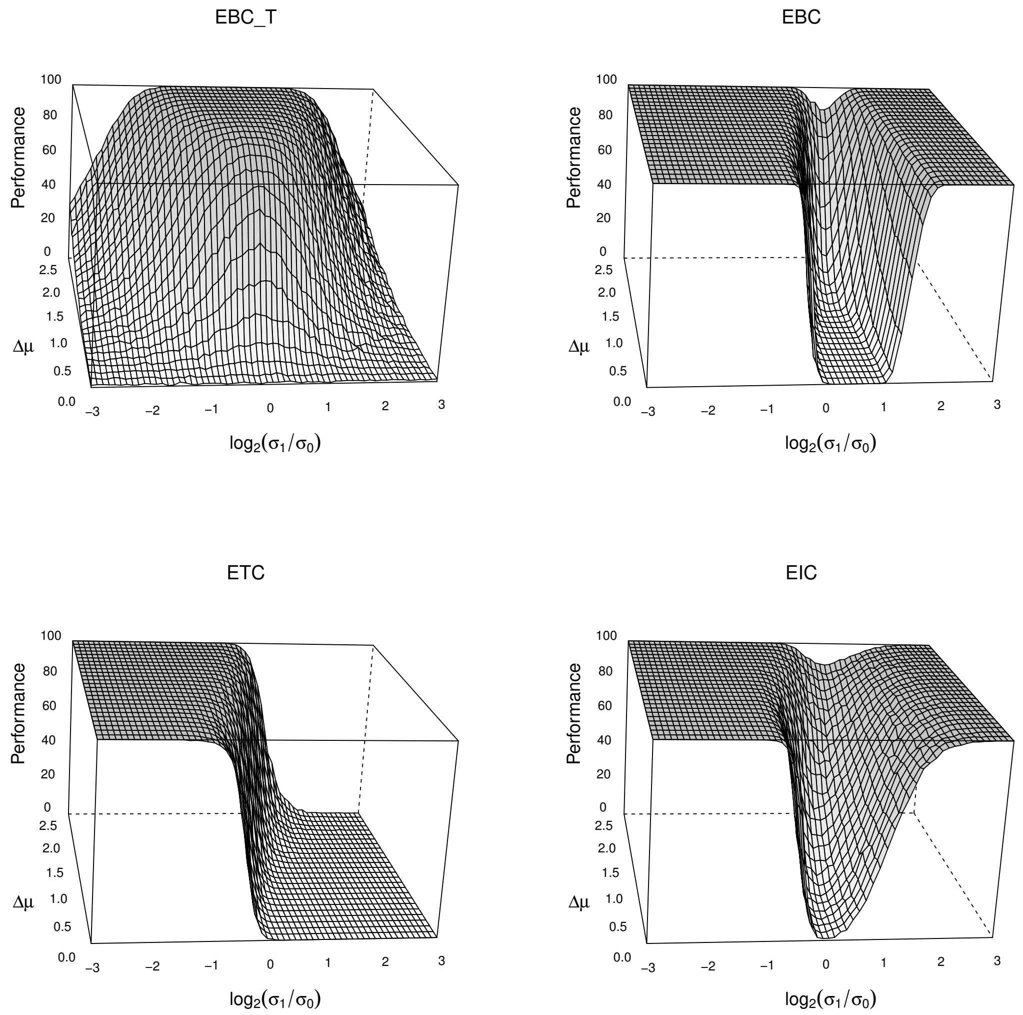


Figure 6.14: This figure illustrates the performance of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 6$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$.

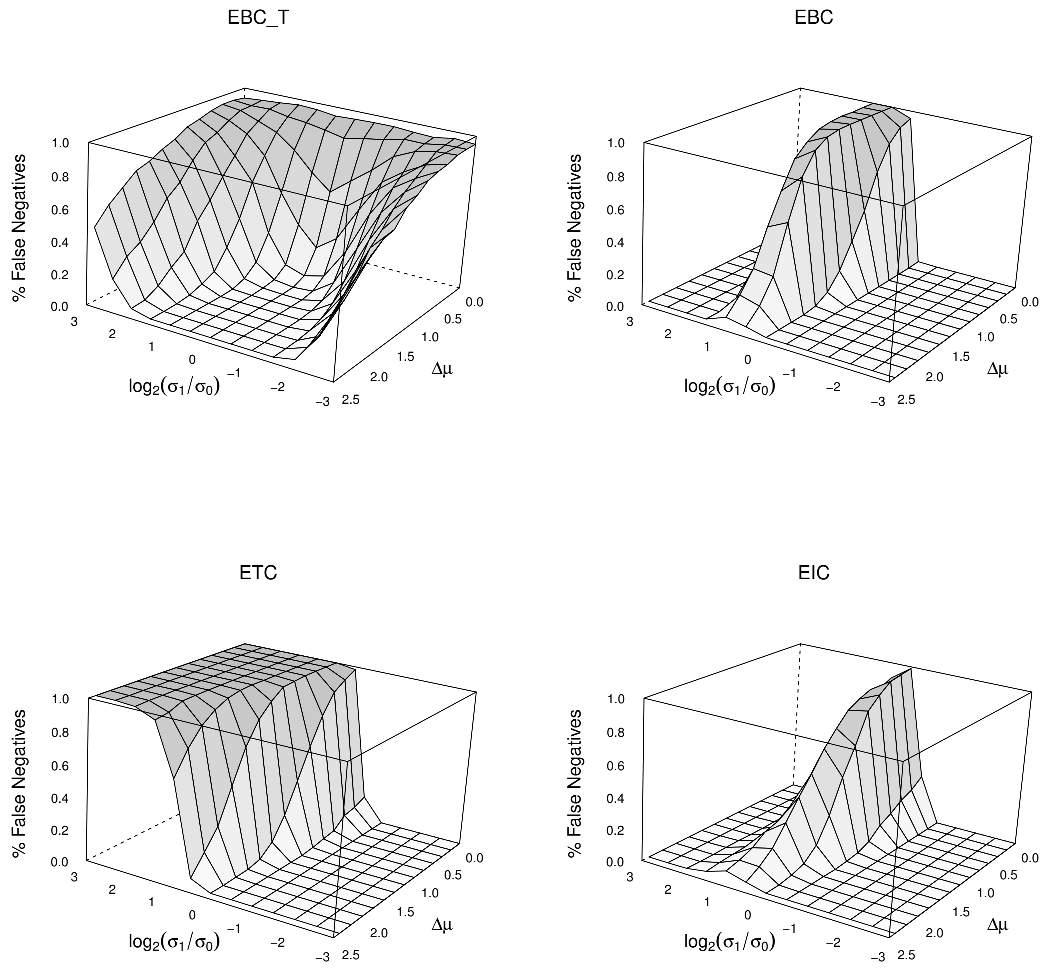


Figure 6.15: This figure illustrates the percentage of false negatives of the filter statistics EBC_T, EBC, ETC, and EIC for varying differences in the central location of the Gaussian class conditionals and varying ratios of their spread. The misclassification costs are set to $c_0 = 1$ and $c_1 = 6$. The EBC filter yields the prediction error of the Bayes classifier for all points except for the hyperplane characterized by $\sigma_1 = \sigma_0$.

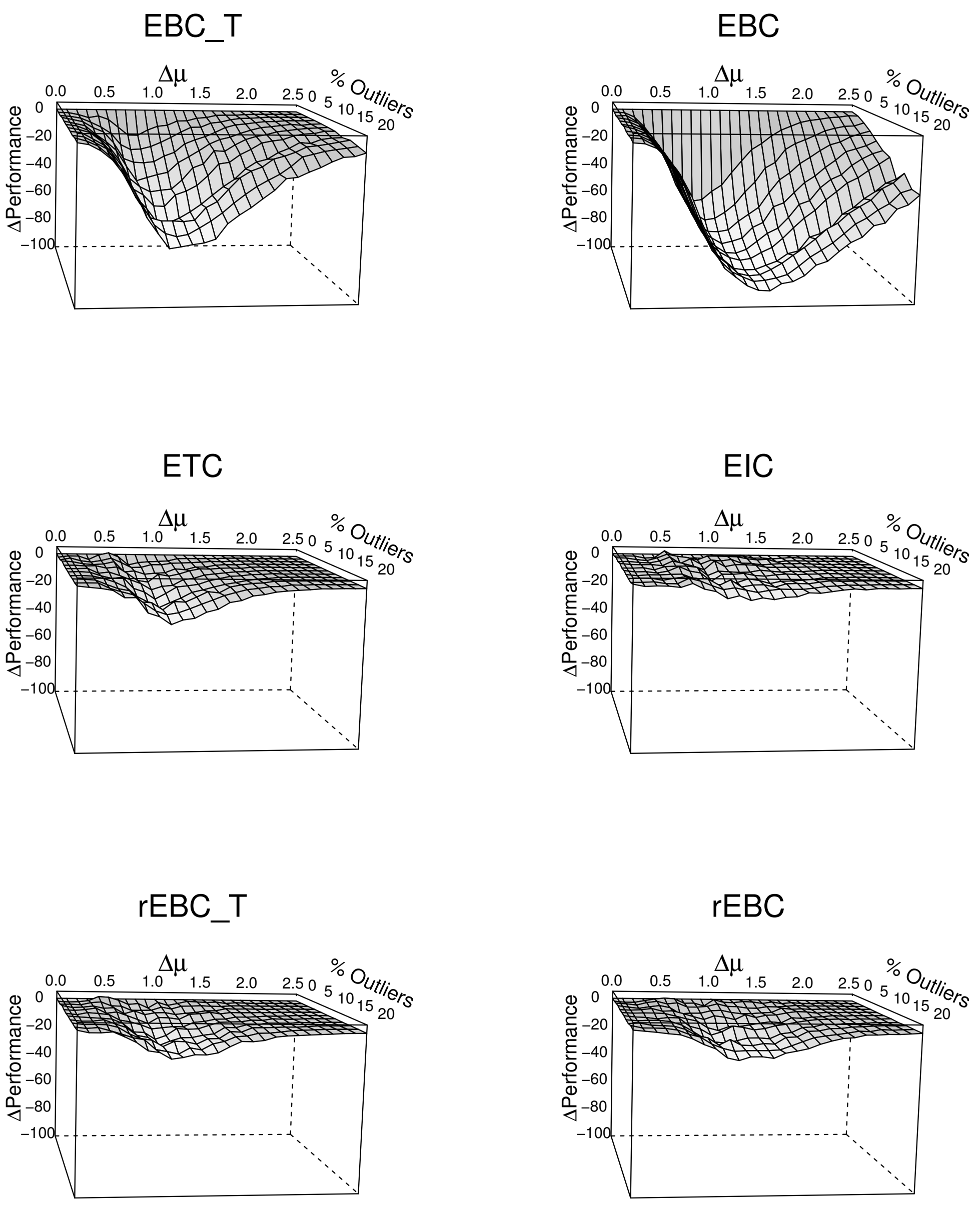


Figure 6.16: Results of Simulation study C for $c_1 = 1$. This experiment analyses the performance of the filters under scrutiny for two Gaussian class conditional distributions that are contaminated with a varying degree of outliers. ΔFP denotes the difference in performance as compared to the simulation study without outliers, all other parameters being equal.

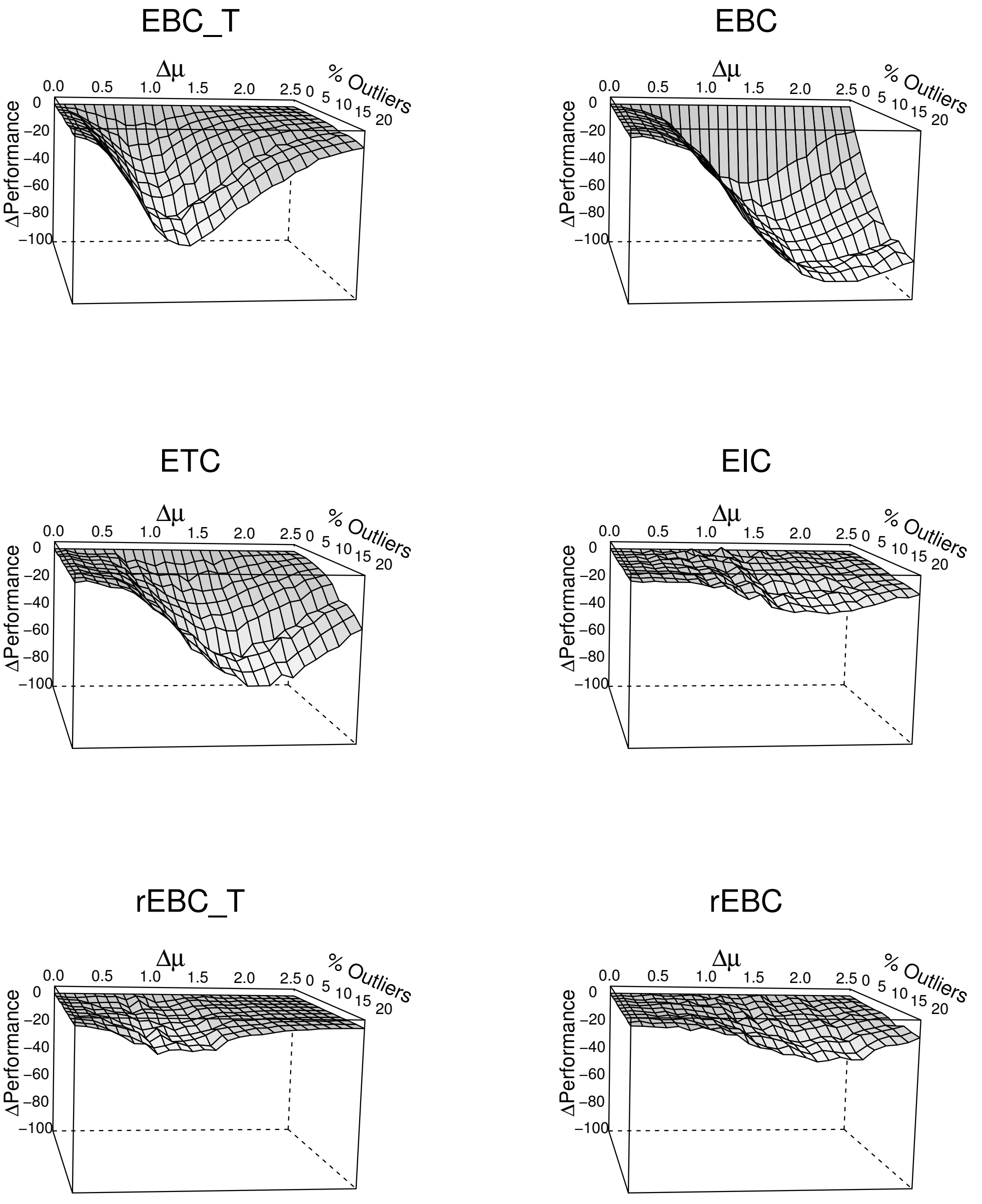


Figure 6.17: Results of Simulation study C for $c_1 = 6$. This experiment analyses the performance of the filters under scrutiny for two Gaussian class conditional distributions that are contaminated with a varying degree of outliers. ΔFP denotes the difference in performance as compared to the simulation study without outliers, all other parameters being equal.

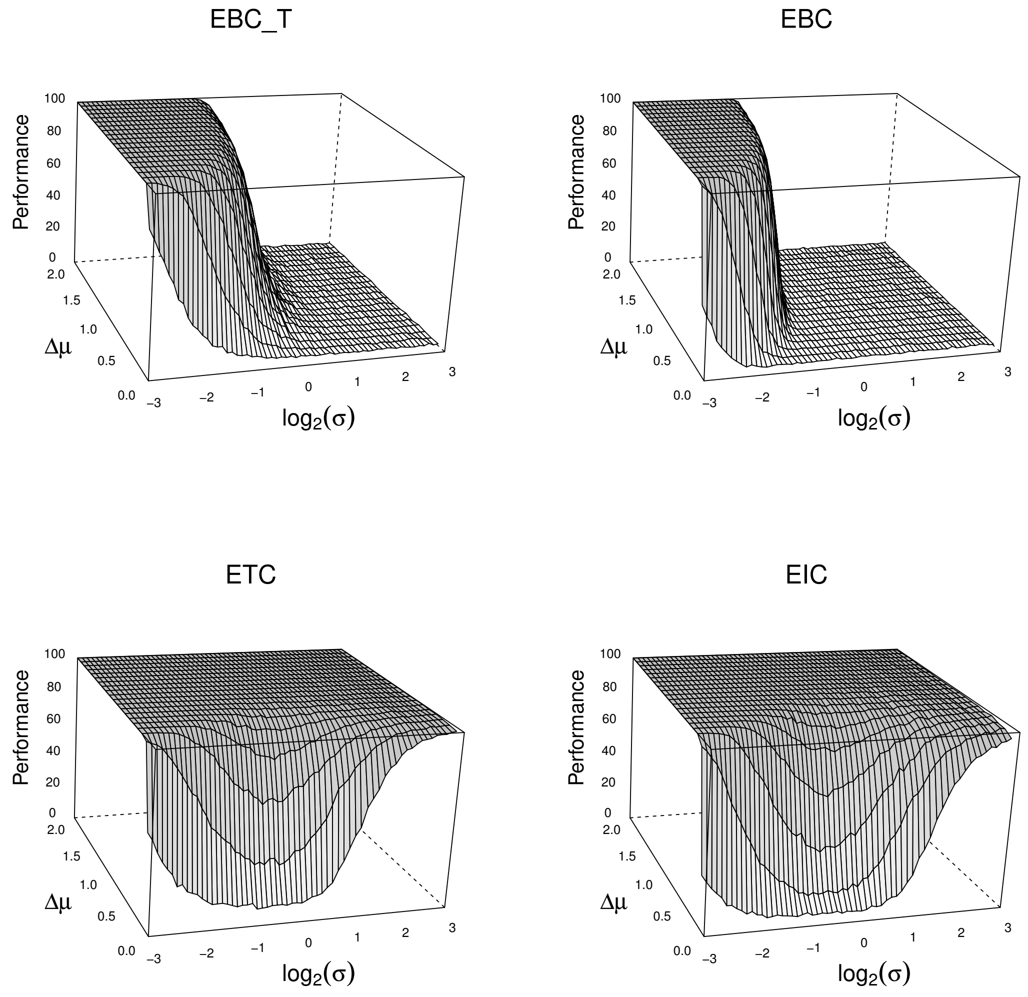


Figure 6.18: Results of Simulation study D for $c_1 = 3$. This experiment analyses the performance of the filter statistics under scrutiny for two lognormal class conditional distributions with varying degrees of skewness.

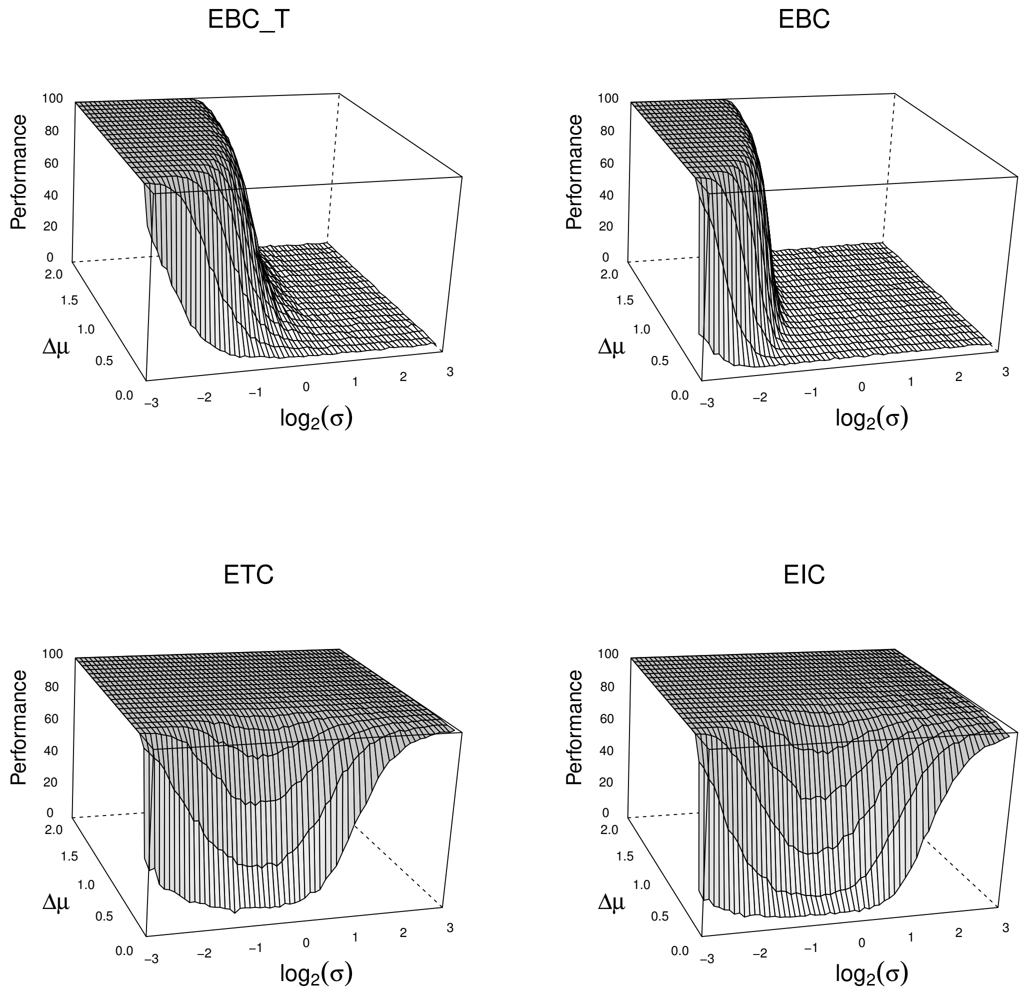


Figure 6.19: Results of Simulation study D for $c_1 = 6$. This experiment analyses the performance of the filter statistics under scrutiny for two lognormal class conditional distributions with varying degrees of skewness.

CHAPTER 7

Real Data Studies

In this section we will finally apply the filters to real data. Generally it is not completely clear what lessons we can expect to learn from real data sets since the performance of a filter depends to a large extent on the data it is applied to. E.g. the presence of outliers will favor one method, skewness another. The characteristics of a data set are usually not known a-priori. The results are, thus, highly domain specific. Since the motivating example for this thesis concerned gene expression data, we will use five famous and publicly available data sets for this analysis. A brief description plus information where to access the data can be found in Section 7.1.

We will limit our analysis to the following two questions: Firstly, what is the benefit of OC-sensitive filtering for a OC-sensitive classification task. We will, thus, compare the performance of the t-test, which stands as a proxy for a variable filter that is non-sensitive to OCs with the performance of EBC_T, EBC, ETC, and EIC. All filters will be applied to the data sets to rank the variables. For the first m top ranked variables, where $m \in \{1, \ldots, 10\}$, models will be fitted with all possible combinations of variables from the subset and the in-sample prediction error will be calculated. The classifier is a multivariate QDA that is OC-sensitive. This approach is equivalent to the simulation study E. The results are depicted in Figure 7.1.

The second question concerns the similarity of the variable filters. To shed some light on this question we will calculate Spearman's rank correlation and generate scatterplots of the ranks of the different filter statistics. Again, this constitutes the same approach as in simulation study E. The results can be found in Tables 7.1 to 7.5 and Figure 7.2.

7.1 Description of Data Sets

Golub et al.

Title: Molecular classification of cancer: class discovery and class prediction by gene expression monitoring.
Description: This data set is probably the most famous and most studied gene expression data set. It analyses the gene expression profiles of human acute myeloid (AML) and acute lymphoblastic leukemias (ALL). The original research is one of the first to show a new approach to cancer classification based on gene expression monitoring by DNA microarrays.
Citation: T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, Oct. 1999
Number of genes: 5147
Number of samples: 72
Diagnostic classes: Acute Lymphoblastic Leukemia (ALL): 47 samples and Acute Myeloid Leukemia (AML): 25 samples.
Platform: Affymetrix HuGeneFL array
Data Source: The `golubEsets` package on Bioconductor.

Mura and De Perrot

Title: Gene expression profiles based on Pulmonary Artery Pressures in Pulmonary Fibrosis.
Description: Identify the gene expression profiles in Pulmonary Fibrosis patients with and without Pulmonary Hypertension.
Citation: M. Mura, M. Anraku, Z. Yun, K. McRae, M. Liu, T. K. Waddell, L. G. Singer, J. T. Granton, S. Keshavjee, and M. de Perrot. Gene expression profiling in the lungs of patients with pulmonary hypertension associated with pulmonary fibrosis. *CHEST Journal*, 141(3):661–673, 2012
Number of genes: 20916
Number of samples: 84
Diagnostic classes: severe PH: 67 samples and without PH: 17 samples
Platform: Affymetrix Human Gene 1.0 ST Array
Data Source: Gene Expression Omnibus (GSE24988)

MacDonald et al.

Title: Expression profiling of medulloblastoma
Description: This publication provides the first insight into the genetic regulation of medulloblastoma metastasis and is the first to suggest a role for PDGFRA and the

RAS/MAPK signaling pathway in medulloblastoma metastasis. Inhibitors of PDGFRA and RAS proteins should therefore be considered for investigation as possible novel therapeutic strategies against medulloblastoma.
Citation: T. J. MacDonald, K. M. Brown, B. LaFleur, K. Peterson, C. Lawlor, Y. Chen, R. J. Packer, P. Cogen, and D. A. Stephan. Expression profiling of medulloblastoma: PDGFRA and the RAS/MAPK pathway as therapeutic targets for metastatic disease. *Nature Genetics*, 29(2):143–152, 2001
Number of genes: 2059
Number of samples: 23
Diagnostic classes: Metastatic and non-metastatic.
Platform: Affymetrix Human Cancer Array
Data Source: Gene Expression Omnibus (GSE468)

Tsukamoto et al.

Title: Clinical Significance of Osteoprotegerin Expression in Human Colorectal Cancer
Description: This study aimed to identify a novel biomarker or a target of treatment for colorectal cancer (CRC). The expression profiles of cancer cells in 148 patients with CRC were examined using laser microdissection and oligonucleotide microarray analysis.
Citation: S. Tsukamoto, T. Ishikawa, S. Iida, M. Ishiguro, K. Mogushi, H. Mizushima, H. Uetake, H. Tanaka, and K. Sugihara. Clinical significance of osteoprotegerin expression in human colorectal cancer. *Clinical Cancer Research*, 17(8):2444–2450, 2011
Number of genes: 54675
Number of samples: 148
Diagnostic classes: Metastatis and non-metastasis
Platform: Affymetrix Human Genome U133 Plus 2.0 Array
Data Source: Gene Expression Omnibus (GSE21510)

Heap et al.

Title: Primary human leucocyte RNA expression of unrelated celiac disease cases and unrelated healthy controls.
Description: The goal of this study was to study the effect of genetic variation on gene expression of untouched primary leucocytes. This expression data is used in conjunction with genome-wide association genotype data.
Citation: G. A. Heap, G. Trynka, R. C. Jansen, M. Bruinenberg, M. A. Swertz, L. C. Dinesen, K. A. Hunt, C. Wijmenga, L. Franke, et al. Complex nature of snp genotype effects on gene expression in primary human leucocytes. *BMC Medical Genomics*, 2(1):1, 2009
Number of genes: 18981
Number of samples: 132
Diagnostic classes: Celiac disease and healthy controls.

Platform: Illumina humanRef-8 v2.0 expression beadchip
Data Source: Gene Expression Omnibus (GSE11501)

7.2 Results

Figure 7.1 depicts the evolution of the EPE of the best model for the m top ranked variables by five different variable filters. The first observation is that there is not one filter that dominates all others in the sense that it always performs better. Building on the lessons learned in the simulation studies, this was not to be expected anyway. The top performing filter in one method can be the worst performer in another depending on the idiosyncrasies of the data set. The obvious conclusion that can be drawn from this observation is that if one is concerned about false negative variables one could apply several filters and join the subsets of top ranked variables.

The range of the filter performance is considerable. In two out of the three data sets the EPE of the best performer is less than half the EPE of the weakest performer. The choice of filter is, thus, not negligible.

As a general impression also from other studies which were not included in this thesis is that the t-test performs mediocre and that it obviously depends of the operating conditions (among other things). The more unequal the misclassification costs are, the weaker its performance.

The ranks for data set 2 (Mura and De Perrot) of five different variable filters are plotted against each other in Figure 7.2. The conclusions that can be drawn from the real data result are the same as for simulation study E. The ranks of the t-test and EBC_T show a very high degree of correlation for the top ranking variables. The corresponding values in Tables 7.1 to 7.5 range from 0.89 to 0.96. These methods can, thus, almost be used as substitutes. The second highest correlation that can be observed is between EBC and ETC and ranges from 0.45 to 0.7.

For the other correlations no clear pattern can be observed. Even though most of the correlation coefficients are positive the variance is extremely high. The correlation of ETC and EIC varies from -0.21 to 0.84, the correlation of EBC and EIC from 0.62 to -0.37. This is further evidence of the fact that the performance of a filter is extremely domain dependent. Even among the class of gene expression data sets the variety of data is so big that no clear conclusions can be drawn as to which filter to use. Only if some aspects of the data are known, e.g. that the data contains strong outliers, preferences for certain methods can be drawn.

The lines in the scatter plots of ETC and EIC are caused by the discrete nature of these filters. There is only a finite number of values the test statistic can take and, thus, there will be a high number of variables sharing the same rank.

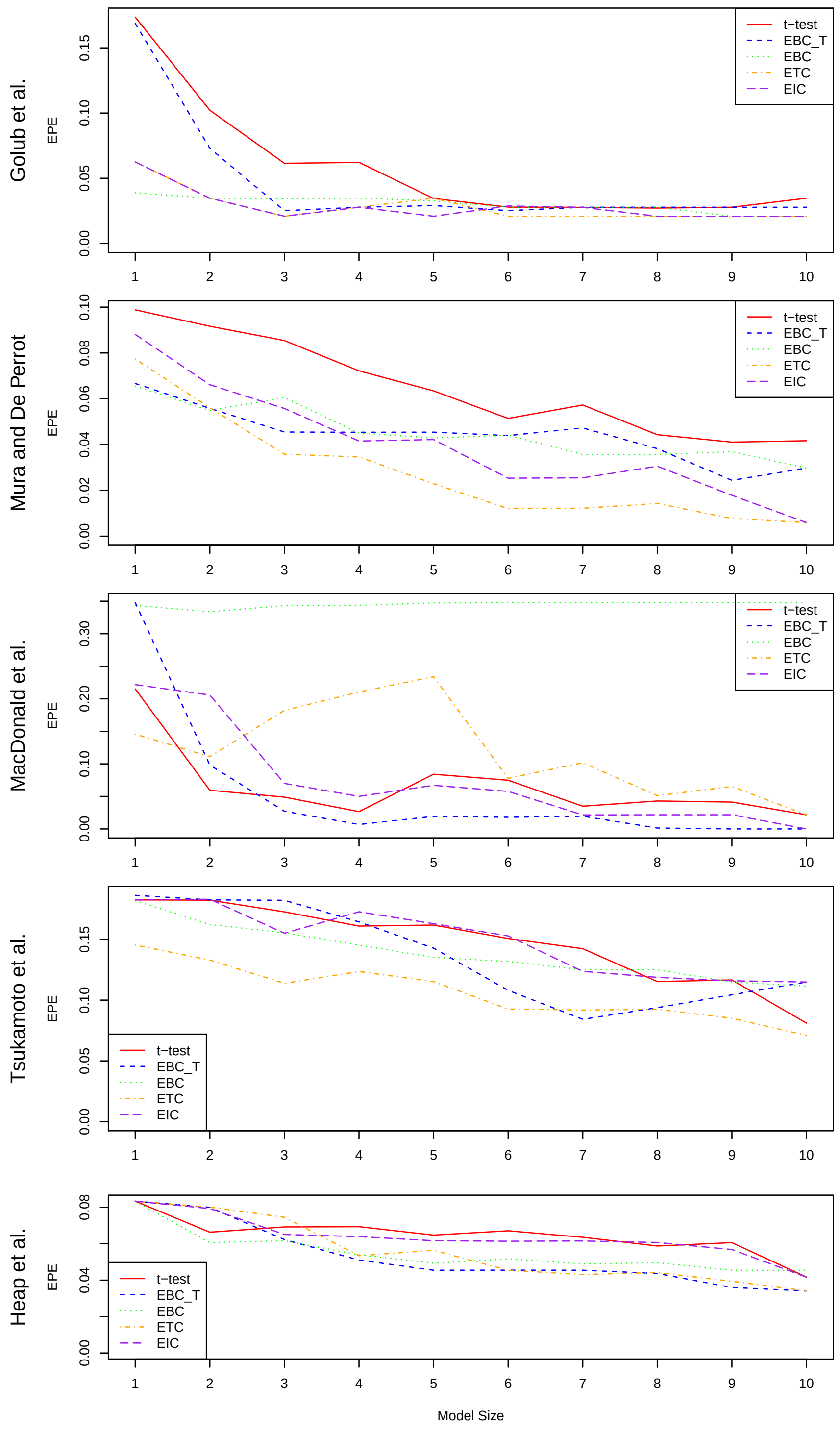


Figure 7.1: depicts the evolution of the EPE of the best model for the m top ranked variables by five different variable filters.

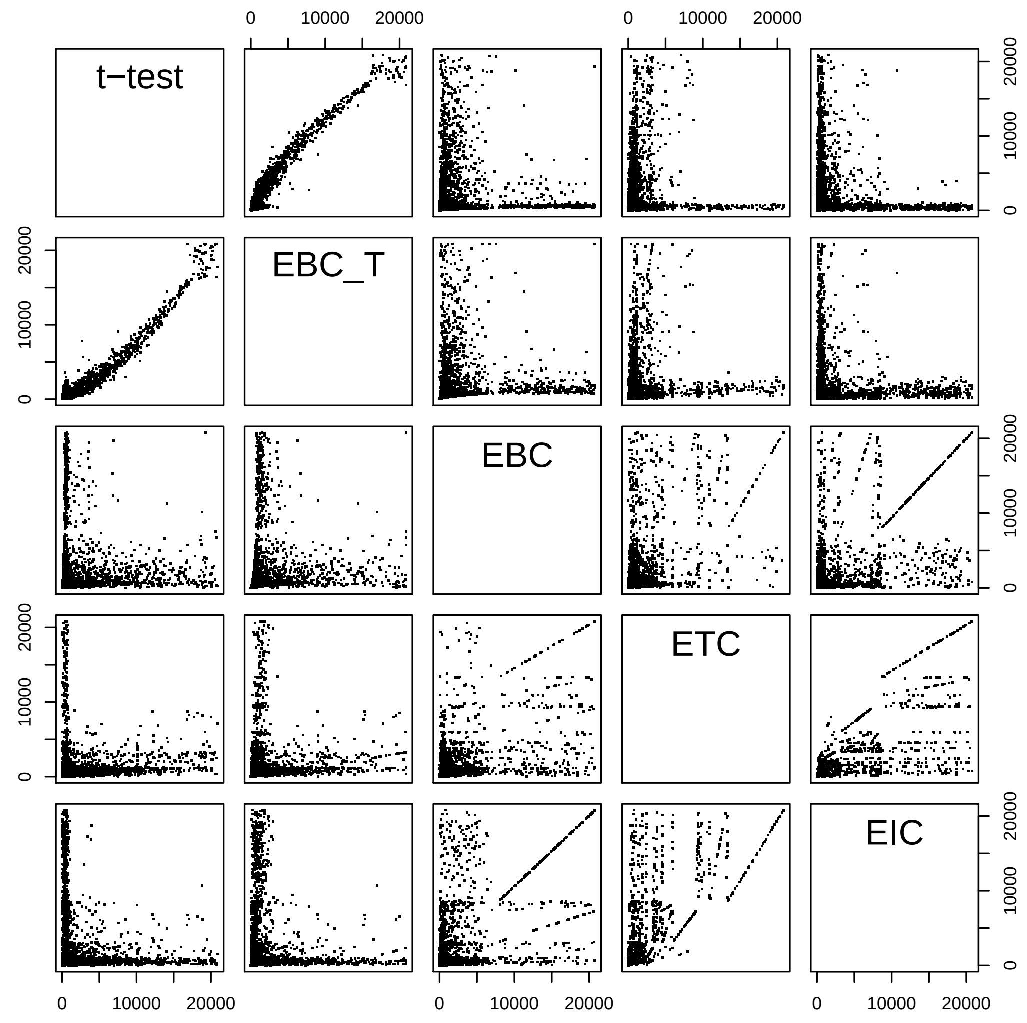


Figure 7.2: plots the ranks of data set 2 (Mura and De Perrot) of five different filters against each other.

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.94	1			
EBC	0.12	0.23	1		
ETC	0.25	0.32	0.58	1	
EIC	0.33	0.36	0.16	0.19	1

Table 7.1: Spearman's rank correlations of the dataset of Golub et al.

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.98	1			
EBC	0.16	0.28	1		
ETC	0.37	0.47	0.67	1	
EIC	0.24	0.33	0.62	0.84	1

Table 7.2: Spearman's rank correlations of the dataset of Mura and De Perrot

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.94	1			
EBC	0.29	0.03	1		
ETC	0.41	0.31	0.45	1	
EIC	0.26	0.39	-0.37	-0.21	1

Table 7.3: Spearman's rank correlations of the dataset of MacDonald et al.

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.89	1			
EBC	0.19	0.23	1		
ETC	0.17	0.21	0.6	1	
EIC	0.31	0.34	0.45	0.3	1

Table 7.4: Spearman's rank correlations of the dataset of Tsukamoto et al.

	t-test	EBC_T	EBC	ETC	EIC
t-test	1				
EBC_T	0.96	1			
EBC	0.26	0.39	1		
ETC	0.27	0.37	0.7	1	
EIC	0.1	0.18	0.51	0.7	1

Table 7.5: Spearman's rank correlations of the dataset of Heap et al.

CHAPTER 8

Conclusions

The starting point of this thesis was the apparent lack of filter methods that are sensitive to the operating conditions. In the introductory chapter, we argued that the neglect of the operating conditions can lead to wrong conclusions, in particular, it can lead to a ranking which ignores the idiosyncrasies of the classification task. We, thus, proposed filtering statistics based on the expected prediction error of a univariate classifier. This statistic can be written as a weighted mean of the false positive and the false negative rates, where the weights are functions of the operating conditions. Any classifiers will do, however, the choice should not be arbitrary since it will generally determine the characteristics of the filtering method. An obvious parametric approach was the Bayesian classifier for a mixture of Gaussian class conditionals, yielding a filter coined *EBC*. In Section 3 we have seen that this classifier is a member of the family of threshold classifiers or of interval classifiers depending on the variances of the class conditional distributions. A filter based on such a parametric model will have similar characteristics as the classifier itself. It relies largely on the parametric assumptions made and any deviation from these will reduce the performance of the filter dramatically. Choosing the classifier is critical and must not be arbitrary. If the class conditionals were known, we could simply choose the classifier that minimizes the prediction error. However, this is rarely the case and would furthermore require a different classifier for every variable. Choosing a non-parametric classifier should remedy this issue. Thus, instead of assuming a parametric family for the class conditionals we will assume that the optimal classifier is a member of the threshold or the interval classifiers. This assumption should hold true for a great number of distributions. Furthermore, it is very popular for medical applications due to its straightforward interpretability, e.g. a fever thermometer. If the measured temperature exceeds a certain critical value the individual is classified as ill. Subsequently, we will fit the model, by finding the optimal classifier among these families. The resulting filters should exhibit the same features as the classifier itself. The simulation studies conducted in Section 6 verify that the filter is robust against outliers or skewness and that the loss in power is negligible in many situations.

These methods exhibit another interesting property. It is possible to obtain the exact finite sample distribution of the test statistic under the null hypothesis of equal class conditional distributions. This distribution is further independent of the class conditional distribution. This allows to apply the null distribution to a mixture of any two distributions, and, thus, constitutes a truly non-parametric method. This distribution is derived in Sections 4.3.2 and 5.3.2. From a practical perspective, this means that the proposed methods can not only be used to rank variables but also to decide how many variables to select based on the data. The proposed filter methods were analyzed both by simulated and real data in Sections 6 and 7. The following conclusions can be drawn: Firstly, the performance of a filter depends strongly on the variables and, thus, no single filter statistic can ever be expected to outperform its competitors in all situations. The performance of a filter is strongly dependent on the domain of the data. If the specific idiosyncrasies of the data are known one can decide on the most promising filter. In all other situations, we would advise to apply several filters and merge the top ranking variables of different filter statistics.

APPENDIX A

Appendix

A.1 Derivation of Equation (2.6) from the Expected Prediction Error

Consider the random variable $(X, Y) : \Omega \mapsto \mathbb{R} \times \{0, 1\}$, where X is continuous and Y denotes the class membership and is, thus, discrete. Y is not observable and is supposed to be predicted by a classifier $\delta : \mathbb{R} \to \{0, 1\}$. In order to evaluate the classifier, we will estimate the expected prediction error defined in (2.4), where $L(\delta(X), Y)$ is a loss function defined in (2.5). If there exists a density $f(x, y)$ for the probability distribution of (X, Y) and it is integrable with respect to the product measure then by Fubini's theorem the integral can be split into iterated integrals, yielding

$$EPE(\delta) = \int_{\mathbb{R}\times\{0,1\}} L(\delta(x), y) f(x, y) d(x, y) = \int_{\mathbb{R}} \int_{\{0,1\}} L(\delta(x), y) f(x, y) dy dx.$$

If we substitute the joint density by the conditional probability and the marginal density and integrate, this yields

$$EPE(\delta) = \int_{\mathbb{R}} c_0 \mathbb{1}_{\delta(x)=1}(x) P(y = 0|x) f_X(x) dx + \int_{\mathbb{R}} c_1 \mathbb{1}_{\delta(x)=0}(x) P(y = 1|x) f_X(x) dx.$$

Using Bayes' theorem we can substitute for $P(y|x)$, yielding

$$\begin{aligned}
EPE(\delta) = &\int_{\mathbb{R}} c_0 \mathbb{1}_{\delta(x)=1}(x) \frac{f_X(x|y=0)P(Y=0)}{f_X(x)} f_X(x)dx + \\
&+ \int_{\mathbb{R}} c_1 \mathbb{1}_{\delta(x)=0}(x) \frac{f_X(x|y=1)P(Y=1)}{f_X(x)} f_X(x)dx = \\
=& c_0(1-\pi_1) \int_{\{x\in\mathbb{R}:\delta(x)=1\}} f_X(x|y=0)dx + c_1\pi_1 \int_{\{x\in\mathbb{R}:\delta(x)=0\}} f_X(x|y=1)dx = \\
=& c_0(1-\pi_1) \underbrace{P(\delta(X)=1|Y=0)}_{\text{false positive rate}} + c_1\pi_1 \underbrace{P(\delta(X)=0|Y=1)}_{\text{false negative rate}}.
\end{aligned}$$

A.2 Implementation of $\widehat{EBC}$

A.2.1 ebc.R

```
#' Expected Loss of the Bayesian Classifier
#'
#' @description The function offers a method to select variables by
    univariate
#' filtering based on the estimated loss of the univariate Bayesian
    Classifer.
#' The statistic requires the parametric assumption that the
    variable consists
#' of a mixture of Gaussian variables.
#'
#' @param class a factor vector indicating the class membership of
    the
#' instances. Must have exactly two levels.
#' @param data a data frame with the variables to filer in columns.
#' @param oc a vector containing three elements. oc[1], the cost of
#' misclassifying a negative instance, oc[2], the cost of
    missclassifying a
#' positive instance, and oc[3], the share of negative instances in
    the
#' population.
#' @param positive a character object indicating the factor label of
     the
#' positive class.
#' @param robust a logical indicating whether a robust estimator of
    the mean
#' and variance of the two classes should be used.
#' @param p.val a logical indicating whether the p-values of ebc
    values under
#' the null hypothesis that both classes are equal should be
    calculated.
#' Currently the null distribution is calculated by permutation.
#' @param adj.method a character string indicating the method with
    which to
#' correct the p-values for multiple testing. See ?p.adjust.
#'
#' @return a list containing three components:
#' \item{ebc}{ a numerical vector containing the etc values for
    every variable
#' of dat.}
#' \item{p.val}{ the corresponding p-values of etc. (optional)}
#' \item{p.val.adj}{ the corresponding adjusted p-values of ebc. (
    optional)}
#'
#' @examples
#' oc <- c(1,3,0.5)
#' class <- factor(c(rep(0, 25), rep(1, 25)), labels=c("neg", "pos")
   )
#' data <- data.frame("var1"=c(rnorm(25, 0, 1/2), rnorm(25, 1, 2)))
#' res <- ebc(class, data, oc, positive="pos", p.val=TRUE)
#' @export

ebc <- function(class, data, positive = levels(class)[1], oc = c(1,
   1, 0.5),
                robust = FALSE,  p.val = TRUE, adj.method = "BH",
                   equalVars = FALSE) {

  # Error handling
  if (is.null(dim(data))) data <- as.data.frame(data)
  error_handling("ebc", class, data, positive, oc, p.val, adj.method
     )

  # Define variables
  p <- ifelse(class(data) == "numeric", 1, ncol(data))
  n <- length(class)
  pos <- class == positive
  neg <- !pos

  # Return objects
  epe <- vector(mode = "numeric", length = p)
  names(epe) <- colnames(data)
  p.value <- vector(length = p)

  # Calculate the prediction error for every variable in the data
     set
  for (i in 1:p) {
    mu1 <- ifelse(robust, median(data[pos, i], na.rm = TRUE),
                          mean(data[pos, i], na.rm = TRUE))
    mu0 <- ifelse(robust, median(data[neg, i], na.rm = TRUE),
                          mean(data[neg, i], na.rm = TRUE))
    sigma1 <- ifelse(robust, mad(data[pos, i], na.rm = TRUE),
                             sd(data[pos, i], na.rm = TRUE))
    sigma0 <- ifelse(robust, mad(data[neg, i], na.rm = TRUE),
                             sd(data[neg, i], na.rm = TRUE))

    if (sigma1 == 0) sigma1 <- 1e-12
    if (sigma0 == 0) sigma0 <- 1e-12


    if (sigma0 == sigma1 | equalVars){ # equal variances

      d.std <- data[,i]
      d.std[pos] <- d.std[pos] - mean(data[pos,i], na.rm = TRUE)
      d.std[neg] <- d.std[neg] - mean(data[neg,i], na.rm = TRUE)
    sigma <- ifelse(robust, mad(d.std, na.rm = TRUE), sd(d.std, na
        .rm = TRUE))

    x <- log(oc[1] / oc[2] * oc[3] / (1 - oc[3])) * sigma^2 / (mu1
        - mu0) + (mu1 + mu0)/2

    epe[i] <- ifelse(mu0 < mu1,
                     oc[3] * oc[1] * (1 - pnorm(x, mu0, sigma)) +
                       (1 - oc[3]) * oc[2] * pnorm(x, mu1, sigma),
                     oc[3] * oc[1] * pnorm(x, mu0, sigma) +
                       (1 - oc[3]) * oc[2] * (1 - pnorm(x, mu1,
                         sigma)))

  } else { # unequal variances

    if (mu0 < mu1) {
      xm <- roots(mu0, mu1, sigma0, sigma1, oc[1], oc[2], oc[3])
          [2]
      xa <- roots(mu0, mu1, sigma0, sigma1, oc[1], oc[2], oc[3])
          [1]
      if (is.na(xm)) {
        epe[i] <- (min((1 - oc[3]) * oc[2], oc[3] * oc[1]))
      } else {
        epe[i] <- ifelse (sigma0 < sigma1,
                          oc[3] * oc[1] * (1 - pnorm(xm, mu0,
                              sigma0) + pnorm(xa, mu0, sigma0)) +
                            (1 - oc[3]) * oc[2] * (pnorm(xm, mu1,
                              sigma1) - pnorm(xa, mu1, sigma1)),
                          oc[3] * oc[1] * (pnorm(xa, mu0, sigma0)
                              - pnorm(xm, mu0, sigma0)) +
                            (1 - oc[3]) * oc[2] * (1 - pnorm(xa,
                              mu1, sigma1) + pnorm(xm, mu1,
                              sigma1)))
      }
    } else {
      xm <- roots(mu0, mu1, sigma0, sigma1, oc[1], oc[2], oc[3])
          [1]
      xa <- roots(mu0, mu1, sigma0, sigma1, oc[1], oc[2], oc[3])
          [2]
      if (is.na(xm)) {
        epe[i] <- (min((1 - oc[3]) * oc[2], oc[3] * oc[1]))
      } else {
        epe[i] <- ifelse (sigma0 < sigma1,
                          oc[3] * oc[1] * (1 - pnorm(xa, mu0,
                              sigma0) + pnorm(xm, mu0, sigma0)) +
                          (1 - oc[3]) * oc[2] * (pnorm(xa, mu1,
                              sigma1) - pnorm(xm, mu1, sigma1)),
                          oc[3] * oc[1] * (pnorm(xm, mu0, sigma0)
                              - pnorm(xa, mu0, sigma0)) +
                                    (1 - oc[3]) * oc[2] * (1 - pnorm(xm, mu1
                                       , sigma1) + pnorm(xa, mu1, sigma1)))
          }
        }
        if (sigma0 == 0 | sigma1 == 0) {
          epe[i] <- NA
          }
        if (is.na(xm)) {
          epe[i] <- (min((1 - oc[3]) * oc[2], oc[3] * oc[1]))
          }
      }
    }


    # generate the null distribution and calculate the p-values
    if (p.val) {

      reps <- 100000
      data.h0 <- matrix(ncol = reps, nrow = length(class), data=rnorm(
          reps*length(class)))
      distr.h0 <- ebc(class = class, data = data.h0, oc = oc, positive
          = positive, p.val = FALSE)$ebc
      min.val <- min(distr.h0)

      # fit empirical cumulative distribution function
      h0.ecdf <- ecdf(distr.h0)

      # fit monotonic spline
      distr.h0.tail <- distr.h0[which(rank(distr.h0)<=200)]
      grid <- seq(0,max(distr.h0.tail), 0.01)
      pts <- which(h0.ecdf(grid)>0)
      h0.spline <- splinefun(c(0,grid[pts]), c(0,h0.ecdf(grid[pts])),
          method = "hyman")
      pval <- function(x) ifelse(x < min.val, h0.spline(x), h0.ecdf(x)
          )
      p.value <- sapply(epe, pval)
      p.val.adj <- p.adjust(p.value, method = adj.method)
    } else {
      p.value <- NULL
      p.val.adj <- NULL
      }

    return(list("ebc" = epe, "p.value" = p.value, "p.value.adj" = p.
        val.adj))

  }


  roots <- function(mu0, mu1, sigma0, sigma1, c0, c1, pi0) {
```

```
  a <- 1 / (2 * sigma0^2) - 1 / (2 * sigma1^2)
  b <- mu1 / sigma1^2 - mu0 / sigma0^2
  c <- mu0^2 / (2 * sigma0^2) - mu1^2 / (2 * sigma1^2) + log(sigma0
      / sigma1) - log(pi0 / (1 - pi0) * c0 / c1)

  if (b^2 - 4 * a * c >= 0) { return(c((- b - sqrt(b^2 - 4 * a * c))
      / (2 * a), (- b + sqrt(b^2 - 4 * a * c)) / (2 * a)))
  } else {
    return(c(NA, NA))
    }
}
```

A.3 Implementation of $\widehat{ETC}$

A.3.1 etc.R

```
#' Expected Loss of the Threshold Classifier.
#'
#' @description The function offers a method to select variables by
    univariate
#' filtering based on the estimated loss of the optimal univariate
    threshold
#' classifer. No parametric assumption about the class conditional
#' distributions is required.
#'
#' @param class a factor vector indicating the class membership of
    the
#' instances. Must have exactly two levels.
#' @param data a data frame with variables in columns.
#' @param oc a vector containing three elements. oc[1], the cost of
#' misclassifying a negative instance, oc[2], the cost of
    missclassifying a
#' positive instance, and oc[3], the share of negative instances in
    the
#' population.
#' @param positive a character object indicating the factor label of
     the
#' positive class.
#' @param p.val a logical indicating whether the p-values of etc
    values under
#' the null hypothesis that both classes are equal should be
    calculated. The
#' exact null distribution is calculated by means of a recursive
    algorithm.
#' @param adj.method a character string indicating the method with
    which to
#' correct the p-values for multiple testing. See ?p.adjust.
#' @param plot a logical. If TRUE a plot of the null distribution
    will be
#' generated.
#'
#' @return a list containing three components:
#' \item{etc}{ a numerical vector containing the etc values for
    every variable
#' of dat.}
#' \item{p.val}{ the corresponding p-values of etc. (optional)}
#' \item{p.val.adj}{ the corresponding adjusted p-values of etc. (
    optional)}
#'
#' @examples
#' oc <- c(1, 3, 0.5)
#' class <- factor(c(rep(0, 25), rep(1, 25)), labels = c("neg", "pos
    "))
#' data <- data.frame("var1" = c(rnorm(25, 0, 1/2), rnorm(25, 1, 2))
    )
#' res <- etc(class, data, positive = "pos", oc, p.val = TRUE)
#'
#' @export

etc <- function(class, data, positive = levels(class)[1], oc = c(1,
    1, 0.5),
                p.val = TRUE, plot = FALSE, adj.method = "BH") {

  # Error handling
  if (is.null(dim(data))) data <- as.data.frame(data)
  error_handling("etc", class, data, positive, oc, p.val, plot, adj.
     method)

  # Define variables
  pos <- class == positive
  neg <- !pos
  p <- ifelse(class(data) == "numeric", 1, ncol(data))
  if (class(data) == "numeric") data <- as.data.frame(data)
  npos <- as.numeric(table(class)[positive])
  nneg <- as.numeric(table(class)[levels(class)[!levels(class) ==
     positive]])

  # Return opjects
  epe <- vector(mode = "numeric", length = p)
  names(epe) <- colnames(data)
  p.value <- vector(length = p)

  # Calculate the prediction error for every variable in the data
     set
  for (i in 1:p) {
    ord <- order(data[, i])
    class.ord <- class[ord]
    feat.ord <- data[ord, i]

    fp1 <- c(0, cumsum(as.numeric(class.ord == positive)))
    fn1 <- c(nneg, nneg - cumsum(as.numeric(!class.ord == positive))
       )
    fp2 <- c(npos, npos - cumsum(as.numeric(class.ord == positive)))
    fn2 <- c(0, cumsum(as.numeric(!class.ord == positive)))

    epe[i] <- min(min(fp1 / npos * oc[2] * (1 - oc[3]) + fn1 / nneg
       * oc[1] * oc[3]),
                  min(fp2 / npos * oc[2] * (1 - oc[3]) + fn2 / nneg
                     * oc[1] * oc[3]))
  }

  # generate the null distribution and calculate the p-values
  if (p.val) {
    ND <- etc.genND(nneg, npos, oc[1], oc[2], oc[3])
    p.val <- cumsum(as.numeric(ND$fav.perm / ND$pos.perm))[match(
       round(epe, 4), round(ND$val, 4))]
    p.value.adj <- p.adjust(p.val, method = adj.method)

    # Generate plot
    if (plot) {
      plot(stats::stepfun(ND$val,
                          c(cumsum(as.numeric(ND$fav.perm / ND$pos.
                             perm)), 1)),
                          main = "Cumulative␣Null␣Distribution␣of␣
                             ETC",
                          xlab = "EPE",
                          ylab = "",
                          pch = ".")
      hist.info <- hist(epe, breaks = ND$val, plot = FALSE)
      points(hist.info$mids[hist.info$count != 0],
             hist.info$counts[hist.info$count != 0] / sum(hist.info$
                counts[hist.info$count != 0]),
             col = "red",
             pch = 20)
      text(hist.info$mids[hist.info$count != 0],
           y = hist.info$counts[hist.info$count != 0] / sum(hist.
              info$counts[hist.info$count != 0]),
           labels = hist.info$counts[hist.info$count != 0],
           col = "red")
    }

  } else {
    p.value <- NULL
    p.value.adj <- NULL
    }

  return(list("etc" = epe, "p.value" = p.val, "p.value.adj" = p.
      value.adj))

}
```

A.3.2 etc_nd.R

```
#' Generate the Null Distribution of the Threshold Classifier.
#'
#' @description The function offers an algorithmic aproach to
    generating the
#' null distribution of the etc classifier under the null hypothesis
     that the
#' distributions of the positive and the negative class are
    identical.
#'
#' @param n0 integer indicating the number of negative instances in
    the
#' sample.
#' @param n1 integer indicating the number of positive instances in
    the
#' sample.
#' @param c0 the cost of misclassifying a negative instance.
#' @param c1 the cost of misclassifying a positive instance.
#' @param pi0 a real number between 0 and 1 indicating the
    percentage of
#' negative instances in the population.
#'
#' @return a list containing three components:
#' \item{val}{ a vector with the number of possible values than etc
    can take.}
#' \item{pos.perm}{ the number of possible permutations for the
    given number of
#' positives and negatives.}
#' \item{fav.perm}{ a vector with the number of favorable
    permutations for every
#' value in val.}
#'
#' @examples
#' etc.genND(25, 27, 1, 3, 0.5)
#'
#' @export

etc.genND <- function(n0, n1, c0, c1, pi0) {

  # error handlig
# generate list with pairs (fp, fn), for which to calculate the
    number of
# favorable permutations
mat <- round(outer(seq(0, n0) / n0 * c0 * pi0, seq(0, n1) / n1 *
    c1 * (1 - pi0), FUN = "+"), 4)
val <- sort(as.vector(mat)[!duplicated(as.vector(mat))])
val <- val[val <= min(c1 * (1 - pi0), c0 * pi0)]
clst <- list()
for (v in val) { clst[[as.character(v)]] <- t(which(mat == v, arr.
    ind = TRUE) - 1) }

# unlist clst
clst.vec <- clst[[1]]
colnames(clst.vec)[1] <- paste0(clst[[1]], collapse = ",")
for (i in 2:length(clst)) {
  clst.vec <- cbind(clst.vec, clst[[i]])
  colnames(clst.vec)[(ncol(clst.vec) - ncol(clst[[i]]) + 1) : ncol
      (clst.vec)] <-
    apply(clst[[i]], 2, function(x) paste0(x, collapse = ","))
}

# calculate the number of possible permutations
pos.perm <- gmp::chooseZ(n0 + n1, n0)

# for every pair in clst calculate the number of favorable
    permutations
fav.perm <- rep(gmp::as.bigz(0), length(val))

posLeft.lst <- posLeft(clst.vec, n0, n1, c0, c1, pi0)
negRght.lst <- negRght(clst.vec, n0, n1, c0, c1, pi0)
posRght.lst <- posRght(clst.vec, n0, n1, c0, c1, pi0)
negLeft.lst <- negLeft(clst.vec, n0, n1, c0, c1, pi0)

fav.perm.vec <- posLeft.lst * negRght.lst + negLeft.lst * posRght.
    lst

# add all permutations of pair (fp, fn) that add up to the same
    cost
cntr <- 1
for (i in 1:length(clst)){
  erg <- gmp::as.bigz(0)

  for (j in 1:ncol(clst[[i]])) {

    erg <- gmp::add.bigz(erg, fav.perm.vec[cntr])
    cntr <- cntr + 1
  }
  fav.perm[i] <- erg
```

```
    }

    return(list("val" = val, "pos.perm" = pos.perm, "fav.perm" = fav.
        perm))
}
```

A.3.3 etc_posL.cpp

```
/* These functions calculate the number of favorable permutations of
    the null distribution
 * of the etc classifier for the positive left side for a given set
    of parameters fp, fn,
 * n0, n1, c0, c1, pi0. posLeft calculates the starting values and
    calls the recursive
 * function posLeft_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

//[[Rcpp::plugins(cpp11)]]
void posLeft_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int& tp, int& fn,
    int& tn, int& fp, double& wght) {

  // memoization
  int arr[3] = {level, start, stop};
  std::vector<int> itm(arr, arr+3);

  if (memo.find(itm) != memo.end()) { // map lookup
    *sums = *sums + memo.at(itm);
  } else {
    if (level == 1) { // recursion base case
      *sums = *sums + start - stop + 1;
      memo.insert ( std::make_pair(itm, start - stop + 1) );
    } else {

      int level_new = level - 1;
      int mnpsr, mxpsr;
      double crt;
      crt = round((fp - level_new + 1) * wght *1e4)/1e4;
      mnpsr = (fabs(ceil(crt) - crt) < 1e-6) ? round(crt + 1) : ceil
          (crt);
      crt = round((tp - fn - mnpsr - (level_new - tn) * wght) * 1e4)
```

```
          / 1e4;
      mxpsr = floor(crt);
      if (mxpsr >= 0) {
        int start_new = (fp + tp) - mnpsr - (fp - level_new);
        int stop_new = std::max(level_new, tp - mnpsr + level_new -
            mxpsr);
        mpz_class bfr(*sums);
        for (int i = start; i >= stop; i--) {
          start_new = std::min(i-1, start_new);
          posLeft_rec(memo, sums, level_new, start_new, stop_new, tp
              , fn, tn, fp, wght);
        }
        mpz_class aftr(*sums);
        memo.insert ( std::make_pair(itm, aftr - bfr) );
      }
    }
  }
}


// [[Rcpp::export]]
SEXP posLeft(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);

  for (int i = 0; i < ncol; i = i + 1) {

  int fp = clst(0,i);
  int fn = clst(1,i);
  int tp = n1 - fn;
  int tn = n0 - fp;
  int n = n0 + n1;
  mpz_class sums(0);

  std::map <std::vector<int>, mpz_class> memo; // create map

  if (fp == 0) {

    sums = (tp == 0) ? 0 : 1;
    perm[i] = sums;

  } else {

    int level = fp;
    double wght = c0 / c1 * n1 / n0 * pi0 / (1 - pi0);
    double crt = wght;
    int mnpsr = (fabs(ceil(crt) - crt) < 1e-6) ? round(crt+1) : ceil
```

```
          (crt);
      crt = round((tp - fn - mnpsr - (level - tn)*wght)*1e4)/1e4;
      int mxpsr = floor(crt);

      if (mxpsr >= 0) {
        int start = fp + tp - mnpsr;
        int stop = std::max(level, tp - mnpsr + level - mxpsr);

        // start recursion
        posLeft_rec(memo, &sums, level, start, stop, tp, fn, tn, fp,
            wght);
        perm[i] = sums;
      }
    }
    memo.clear(); // delete map
    }
    return(wrap(perm));
}
```

A.3.4 etc_posR.cpp

```
/* These functions calculate the number of favorable permutations of
     the null distribution
 * of the etc classifier for the positive right side for a given set
      of parameters fp, fn,
 * n0, n1, c0, c1, pi0. posRght calculates the starting values and
     calls the recursive
 * function posRght_rec.
 */

#include <wrap.hpp>
#include <map>
#include <Rcpp.h>
#include <math.h>
#include <gmpxx.h>


using namespace Rcpp;

//[[Rcpp::plugins(cpp11)]]

void posRght_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int& tp, int& fn,
    int& tn, int& fp, double& wght) {

  // memoization
  int arr[3] = {level, start, stop};
  std::vector<int> itm(arr, arr+3);
  if (memo.find(itm) != memo.end()) { // map lookup
    *sums = *sums + memo.at(itm);
  } else {
    if (level == 1){ // recursion base case
      *sums = *sums + start - stop + 1;
      memo.insert ( std::make_pair(itm, start - stop + 1) );
    } else {
      int level_new = level - 1;
      int mnpsr, mxpsr;
      double crt;
      crt = round((fp - level_new + 1) * wght *1e4)/1e4;
      mnpsr = (fabs(ceil(crt)) - crt < 1e-6) ? round(crt) : ceil(crt
          );
      crt = round((tp - fn - mnpsr - (level_new - tn) * wght) * 1e4)
          / 1e4;
      mxpsr = (fabs(floor(crt) - crt) < 1e-6) ? round(crt - 1) :
          floor(crt);
      if (mxpsr >= 0) {

        int start_new = (fp + tp) - mnpsr - (fp - level_new);
        int stop_new = std::max(level_new, tp - mnpsr + level_new -
            mxpsr);
        mpz_class bfr(*sums);
        for (int i = start; i >= stop; i--) {
          start_new = std::min(i-1, start_new);
          posRght_rec(memo, sums, level_new, start_new, stop_new, tp
              , fn, tn, fp, wght);
        }
        mpz_class aftr(*sums);
        memo.insert ( std::make_pair(itm, aftr - bfr) );
      }
    }
  }
}


// [[Rcpp::export]]
SEXP posRght(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);

  for (int i = 0; i < ncol; i = i + 1) {

    int fp = clst(0,i);
    int fn = clst(1,i);
    int tp = n1 - fn;
    int tn = n0 - fp;
```

```
      int n = n0 + n1;
      mpz_class sums(0);
      double crt;

      std::map <std::vector<int>, mpz_class> memo; // create map

      if (fp == 0) {

        sums = (tp == 0) ? 0 : 1;
        perm[i] = sums;

      } else {

        int level = fp;
        int mnpsr, mxpsr;
        double wght = c0 / c1 * n1 / n0 * pi0 / (1 - pi0);
        double crt;
        crt =  wght;
        mnpsr = (fabs(ceil(crt)) - crt < 1e-6) ? round(crt) : ceil(crt
            );
        crt = round((tp - fn - mnpsr - (level - tn) * wght) * 1e4) / 1
            e4;
        mxpsr = (fabs(floor(crt) - crt) < 1e-6) ? round(crt - 1) :
            floor(crt);

        if (mxpsr >= 0) {
          int start = (fp + tp) - mnpsr;
          int stop = std::max(level, tp - mnpsr + level - mxpsr);

          // start recursion
          posRght_rec(memo, &sums, level, start, stop, tp, fn, tn, fp,
              wght);
          perm[i] = sums;
        }
      }
      memo.clear(); // delete map
    }
    return(wrap(perm));
  }
```

A.3.5 etc_negL.cpp

```
/* These functions calculate the number of favorable permutations of
     the null distribution
 * of the etc classifier for the positive left side for a given set
     of parameters fp, fn,
 * n0, n1, c0, c1, pi0. negLeft calculates the starting values and
     calls the recursive
 * function negLeft_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <gmp.h>
#include <Rcpp.h>
#include <math.h>


using namespace Rcpp;

//[[Rcpp::plugins(cpp11)]]
void negLeft_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int& tp, int& fn,
    int& tn, int& fp, double& wght) {

  // memoization
  int arr[3] = {level, start, stop};
  std::vector<int> itm(arr, arr+3);

  if (memo.find(itm) != memo.end()) { // map lookup
    *sums = *sums + memo.at(itm);
  } else {
    if (level == 1) { // recursion base case
      *sums = *sums + start - stop + 1;
      memo.insert ( std::make_pair(itm, start - stop + 1) );
    } else {

      int level_new = level - 1;
      int mnpsr, mxpsr;
      double crt;
      crt = round((fn - level_new + 1) * wght *1e4)/1e4;
      mnpsr = (fabs(ceil(crt) - crt) < 1e-6) ? round(crt+1) : ceil(
          crt);
      crt = round( (tn - fp - mnpsr - (level_new - tp) * wght) * 1e4
           ) / 1e4;
      mxpsr = (fabs(floor(crt) - crt) < 1e-6) ? round(crt - 1) :
          floor(crt);
      if (mxpsr >= 0) {
        int start_new = (fn + tn) - mnpsr - (fn - level_new);
        int stop_new = std::max(level_new, tn - mnpsr + level_new -
            mxpsr);
        mpz_class bfr(*sums);
        for (int i = start; i >= stop; i--) {
          start_new = std::min(i-1, start_new);
          negLeft_rec(memo, sums, level_new, start_new, stop_new, tp
              , fn, tn, fp, wght);
        }
```

```
          mpz_class aftr(*sums);
          memo.insert ( std::make_pair(itm, aftr - bfr) );

        }
      }
    }
}


// [[Rcpp::export]]
SEXP negLeft(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);

  for (int i = 0; i < ncol; i = i + 1) {

    int fp = clst(0,i);
    int fn = clst(1,i);
    int tp = n1 - fn;
    int tn = n0 - fp;
    int n = n0 + n1;
    mpz_class sums(0);
    double crt;

    std::map <std::vector<int>, mpz_class> memo; // create map

    if (fn == 0) {

      sums = (tn == 0) ? 0 : 1;
      perm[i] = sums;

    } else {

      int level = fn;
      int mnpsr, mxpsr;
      double crt;
      double wght = c1 / c0 * (tn + fp) / (tp + fn) * (1 - pi0) /
          pi0;
      crt = wght;
      mnpsr = (fabs(ceil(crt) - crt) < 1e-6) ? round(crt + 1) : ceil
          (crt);
      crt = round( (tn - fp - mnpsr - (level - tp) * wght) * 1e4 ) /
          1e4;
      mxpsr = (fabs(floor(crt) - crt) < 1e-6) ? round(crt - 1) :
          floor(crt);

      if (mxpsr >= 0) {
        int start = (fn + tn) - mnpsr - (fn - level);
        int stop = std::max(level, tn - mnpsr + level - mxpsr);

        // start recursion
        negLeft_rec(memo, &sums, level, start, stop, tp, fn, tn, fp,
            wght);
        perm[i] = sums;
      }
    }
    memo.clear(); // delete map
  }
  return(wrap(perm));
}
```

A.3.6 etc_negR.cpp

```
/* These functions calculate the number of favorable permutations of
     the null distribution
 * of the etc classifier for the negative right side for a given set
      of parameters fp, fn,
 * n0, n1, c0, c1, pi0. negRght calculates the starting values and
     calls the recursive
 * function negRght_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

//[[Rcpp::plugins(cpp11)]]
void negRght_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int& tp, int& fn,
    int& tn, int& fp, double& wght) {

  // memoization
  int arr[3] = {level, start, stop};
  std::vector<int> itm(arr, arr+3);

  if (memo.find(itm) != memo.end()) { // map lookup
    *sums = *sums + memo.at(itm);
  } else { // calculate key value
    if (level == 1) { // recursion base case
      *sums = *sums + start - stop + 1;
      memo.insert ( std::make_pair(itm, start - stop + 1) );
    } else {
        int level_new = level - 1;
        int mnpsr, mxpsr;
        double crt;
        crt = round((fn - level_new + 1) * wght *1e4)/1e4;
        mnpsr=ceil(crt);
        crt = round( (tn - fp - mnpsr - (level_new - tp) * wght) * 1e4
            ) / 1e4;
        mxpsr = floor(crt);
        if (mxpsr >= 0) {
          int start_new = (fn + tn) - mnpsr - (fn - level_new);
          int stop_new = std::max(level_new, tn - mnpsr + level_new -
              mxpsr);
          mpz_class bfr(*sums);
          for (int i = start; i >= stop; i--) {
            start_new = std::min(i-1, start_new);
            negRght_rec(memo, sums, level_new, start_new, stop_new, tp
                , fn, tn, fp, wght);
          }
          mpz_class aftr(*sums);
          memo.insert ( std::make_pair(itm, aftr - bfr) );

        }
      }
    }
  }


  // [[Rcpp::export]]
  SEXP negRght(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
      double c1, double pi0) {

    int ncol = clst.ncol();
    std::vector <mpz_class> perm(ncol);

    for (int i = 0; i < ncol; i = i + 1) {

      int fp = clst(0,i);
      int fn = clst(1,i);
      int tp = n1 - fn;
      int tn = n0 - fp;
      int n = n0 + n1;
      mpz_class sums(0);
      double crt;

      std::map <std::vector<int>, mpz_class> memo; // create map

      if (fn == 0) {

        sums = 1;
        perm[i] = sums;

      } else {

        int level = fn;
        double wght = c1 / c0 * n0 / n1 * (1 - pi0) / pi0;
        crt = wght;
        int mnpsr = (fabs(ceil(crt) - crt) < 1e-9) ? round(crt) : ceil
            (crt);
        crt = (tn - fp - mnpsr - (level - tp) * wght);
        int mxpsr = (fabs(floor(crt) - crt) < 1e-9) ? round(crt) :
            floor(crt);

        if (mxpsr >= 0) {
          int start = (fn + tn) - mnpsr;
          int stop = std::max(level, tn - mnpsr + level - mxpsr);

          // start recursion
          negRght_rec(memo, &sums, level, start, stop, tp, fn, tn, fp,
               wght);
          perm[i] = sums;
        }
      }
      memo.clear(); // delete map
    }
    return(wrap(perm));
}
```

A.4 Implementation of $\widehat{EIC}$

A.4.1 eic.R

```
#' Expected Loss of the Interval Classifier.
#'
#' @description The function offers a method to select variables by
    univariate
#' filtering based on the estimated loss of the optimal univariate
    interval
#' classifer. No parametric assumption about the class conditional
#' distributions is required.
#'
#' @param class a factor vector indicating the class membership of
    the
#' instances. Must have exactly two levels.
#' @param data a data frame with variables in columns.
#' @param oc a vector containing three elements. oc[1], the cost of
#' misclassifying a negative instance, oc[2], the cost of
    missclassifying a
#' positive instance, and oc[3], the share of negative instances in
    the
#' population.
#' @param positive a character object indicating the factor label of
      the
#' positive class.
#' @param p.val a logical indicating whether the p-values of etc
    values under
#' the null hypothesis that both classes are equal should be
    calculated. The
#' exact null distribution is calculated by means of a recursive
    algorithm.
#' @param adj.method a character string indicating the method with
    which to
#' correct the p-values for multiple testing. See ?p.adjust.
#' @param plot a logical. If TRUE a plot of the null distribution
    will be
#' generated.
#'
#' @return a list containing three components:
#' \item{eic}{ a numerical vector containing the etc values for
    every variable
#' of dat.}
#' \item{p.val}{ the corresponding p-values of etc. (optional)}
#' \item{p.val.adj}{ the corresponding adjusted p-values of etc. (
    optional)}
#'
#' @examples
#' oc <- c(1,3,0.5)
#' class <- factor(c(rep(0, 25), rep(1, 25)), labels=c("neg", "pos")
    )
#' data <- data.frame("var1"=c(rnorm(25, 0, 1/2), rnorm(25, 1, 2)))
#' res <- eic(class, data, oc, positive="pos", p.val=TRUE)
#'
#' @export


eic <- function(class, data, positive = levels(class)[1], oc = c(1,
    1, 0.5),
                p.val = TRUE, plot = FALSE, adj.method = "BH") {

  # Error handling
  if (is.null(dim(data))) data <- as.data.frame(data)
  error_handling("eic", class, data, positive, oc, p.val, plot, adj.
      method)

  # Define variables
  p <- ifelse(class(data) == "numeric", 1, ncol(data))
  n1 <- as.numeric(table(class)[positive])
  n0 <- length(class) - n1
  n <- n0 + n1
  d1 <- - oc[2] / n1 * (1 - oc[3])
  d0 <- oc[1] / n0 * oc[3]

  # Return objects
  epe <- vector(mode="numeric", length = p)
  names(epe) <- colnames(data)
  p.value <- vector(length = p)

  # Calculate the prediction error for every variable in the data
      set

  # order the class labels according to the ranks of the variables
  data.ord <- matrix(nrow = length(class), data = as.numeric(class)[
      apply(data, 2, order)])

  for (i in 1:p) {
    min.val <- min(oc[2] * (1 - oc[3]), oc[1] * oc[3])

    for (j in 1:n) { # positive interval
      row.min <- min(cumsum( c(oc[2] * (1 - oc[3]), c(d0,d1)[data.
          ord[j:n, i]]) ))
      min.val <- min(min.val, row.min)
    }

    for (j in 1:n) { # negative interval
      row.min <- min(cumsum(c(oc[1] * oc[3], c(-d0, -d1)[data.ord[j:
          n, i]] ) ))
          min.val <- min(min.val, row.min)
        }

        epe[i] <- min.val

      }

      # Generate the null distribution and calculate the p-values

      if (p.val) {
        ND <- eic.genND(n0, n1, oc[1], oc[2], oc[3], data, class,
            positive)
        p.val <- cumsum(as.numeric(ND$fav.perm/ND$pos.perm))[match(round
            (epe,4), round(ND$val,4))]

        if (plot) {
          plot(stats::stepfun(ND$val, c(cumsum(as.numeric(ND$fav.perm /
              ND$pos.perm)), 1)),
                main = "Cumulative␣Null␣Distribution␣of␣ETC", xlab = "EPE
                    ", ylab = "", pch = ".")
          hist.info <- hist(epe, breaks = ND$val, plot = FALSE)
          points(hist.info$mids[hist.info$count != 0], hist.info$counts[
              hist.info$count != 0] / sum(hist.info$counts[hist.info$
              count != 0]), col = "red", pch = 20)
          text(hist.info$mids[hist.info$count != 0], y = hist.info$
              counts[hist.info$count != 0] / sum(hist.info$counts[hist.
              info$count != 0]), labels = hist.info$counts[hist.info$
              count != 0], col = "red")

          p.val.adj <- ifelse(is.null(adj.method), NULL, p.adjust(p.val,
               method = adj.method))
        }

      } else {
        p.val <- NULL
        p.val.adj <- NULL
      }

      return(list("eic" = epe, "p.value" = p.val, "p.value.adj" = p.val.
          adj))

    }
```

A.4.2 `eic_nd.R`

```
#' Generate the Null Distribution of EIC.
#'
#' @description The function offers a heuristic aproach to
    generating the null
```

```
#' distribution of the eic classifier under the null hypothesis that
    the
#' distributions of the positive and the negative class are
   identical.
#'
#' @param n0 an integer indicating the number of negative instances
   in the
#' sample.
#' @param n1 an integer indicating the number of positive instances
   in the
#' sample.
#' @param c0 the cost of misclassifying a negative instance.
#' @param c1 the cost of misclassifying a positive instance.
#' @param pi0 a real number between 0 and 1 indicating the
   percentage of
#' negative instances in the population.
#'
#' @return a list containing three components:
#' \item{val}{ a vector with the number of possible values than eic
   can take.}
#' \item{pos.perm}{ the number of possible permutations for the
   given number of
#' positives and negatives.}
#' \item{fav.perm}{ a vector with the number of favorable
   permutations for
#' every value in val.}
#'
#' @examples
#' eic.genND(25, 27, 1, 3, 0.5)
#'
#' @export

eic.genND <- function(n0, n1, c0, c1, pi0, data, class, positive,
    plot = FALSE) {

  # generate list with pairs (fp, fn), for which to calculate the
      favorable permutations
  mat <- round( outer( seq(0, n0) / n0 * c0 * pi0, seq(0, n1) / n1 *
      c1 * (1 - pi0), FUN = "+"), 4)
  val <- sort(as.vector(mat)[!duplicated(as.vector(mat))])
  val <- val[val <= min(c1 * (1 - pi0), c0 * pi0)]

  # generate list with pairs (fp, fn), for which to calculate the
      favorable permutations
  clst <- list()
  for (v in val) { clst[[as.character(v)]] <- t(which(mat == v, arr.
      ind = TRUE) - 1) }

  # select only those cost values where all pairs of (fp, fn) do not
```

```
    exceed floor (min(n0, n1) / 2)
sel <- sapply(clst, function(y) all(apply(y, 2, function(x) all(x
   <= floor(min(n0,n1) / 2 ) ))))
clst <- clst[sel]

# unlist clst
clst.vec <- clst[[1]]
colnames(clst.vec)[1] <- paste0(clst[[1]], collapse=",")
for (i in 2:length(clst)) {
  clst.vec <- cbind(clst.vec, clst[[i]])
  colnames(clst.vec)[(ncol(clst.vec) - ncol(clst[[i]]) + 1) : ncol
      (clst.vec)] <- apply(clst[[i]], 2, function(x) paste0(x,
      collapse=","))
}

# number of possible permutations
pos.perm <- gmp::chooseZ(n0 + n1, n0)

# number of favorable permutations for every value
fav.perm <- rep(gmp::as.bigz(0), length(val))

# calculate the exact number of favorable permutation for the
    first floor(min(n1,n0)/2) values
posInt.lst <- posInt(clst.vec, n0, n1, c0, c1, pi0)
negInt.lst <- negInt(clst.vec, n0, n1, c0, c1, pi0)
posComp.lst <- posComp(clst.vec, n0, n1, c0, c1, pi0)
negComp.lst <- negComp(clst.vec, n0, n1, c0, c1, pi0)
fav.perm.vec <- posInt.lst * negComp.lst + posComp.lst * negInt.
    lst

cntr <- 1

for (i in 1:(floor(min(n0, n1) / 2))){
  erg <- gmp::as.bigz(0)

  for (j in 1:ncol(clst[[i]])) {

   erg <- gmp::add.bigz(erg, fav.perm.vec[cntr])
    cntr <- cntr + 1
  }
  fav.perm[i] <- erg
}

ind.count <- which(fav.perm != 0)


## ESTIMATE THE NUMBER OF FAVORABLE PERMUTATIONS BY MEANS OF
    RANDOM PERMUTATIONS
# for all values that exhibit more than 5% of overall permutations
```

```
reps <- 20000  # number of random permutations
fav.perm.rnd <- rep(0, length(val))
names(fav.perm.rnd) <- val
dt <- matrix(nrow = nrow(data), ncol = reps)
for (l in 1:ncol(dt)) { dt[,l] <- sample(data[,sample(ncol(data)
    ,1)]) } # fill the matrix with randomly permuted vectors
cvals <- eic(class, dt, c(c0, c1, pi0), positive = positive, p.val
    = FALSE, plot = FALSE, adj.method = NULL)

for (i in 1:length(val)) { fav.perm.rnd[i] <- sum(round(cvals$eic,
    4) == round(val[i], 4) ) }

ind.perm <- fav.perm.rnd >= 0.01 * reps
fav.perm.rnd <- round(fav.perm.rnd * as.numeric(pos.perm / reps))
fav.perm.rnd[!ind.perm] <- 0
fav.perm.rnd[ind.count] <- 0
fav.perm[fav.perm.rnd != 0] <- fav.perm.rnd[fav.perm.rnd != 0]


## ESTIMATE THE NUMBER OF FAVORABLE PERMUTATIONS FOR ALL OTHER
    VALUES BY MEANS OF INTRAPOLATION OR EXTRAPOLATION

# for those values between the maximal value of the exact counting
    schema and the minimal value of the permutation
if (max(ind.count) < min(which(ind.perm)) ) {

  val.r.imp <- c(ind.count[(length(ind.count) - 2) : length(ind.
      count)], which(ind.perm)[c(1,2,3)])
  val.2.imp <- (max(ind.count) + 1):(min(which(ind.perm))-1)
  ptsSpline.mid <- spline(val[val.r.imp], fav.perm[val.r.imp],
      xout = val[val.2.imp], method = "hyman")

  # plot
  #ptsSpline.plt <- spline(val[val.r.imp], fav.perm[val.r.imp], n
      = 200, method = "hyman")
  #plot(ptsSpline.plt, type = "l")
  #points(val[val.r.imp], fav.perm[val.r.imp])
  #points(val[val.2.imp], ptsSpline.mid$y, col="red")

  fav.perm[val.2.imp] <- ptsSpline.mid$y

}

# for those values at the right end

if (sum(fav.perm == 0) > 1) {

  if (sum(fav.perm) < pos.perm) {
        val.ext <- fav.perm == 0
        # intrapolate the cumulative density
        fav.perm.cum <- cumsum(fav.perm)
        fav.perm.cum[val.ext] <- NA
        fav.perm.cum[length(fav.perm.cum)] <- pos.perm
        val.r.imp <- c(which(ind.perm)[(sum(ind.perm)-2):sum(ind.perm)
            ], length(val))
        val.2.imp <- (max(which(ind.perm)) + 1):(length(val) - 1)
        ptsSpline.rght <- spline(val[val.r.imp], fav.perm.cum[val.r.
            imp], xout = val[val.2.imp], method = "hyman")
        fav.perm.cum[val.2.imp] <- round(ptsSpline.rght$y)

        dfs <- as.numeric(fav.perm.cum[c(val.2.imp, length(val))] -
            fav.perm.cum[c(val.2.imp[1] -1) : max(val.2.imp)])
        fav.perm[ (max(which(ind.perm)) + 1) : length(val)] <- sapply(
            dfs, function(z) max(0,z))

      } else {
        # round bigq not implemented yet
        #fav.perm[-ind.count] <- round(fav.perm[-ind.count] * (pos.
            perm - sum(fav.perm[ind.count])) / sum(fav.perm[-ind.count
            ]))
      }
    } else if (sum(fav.perm == 0) == 1) {
      fav.perm[fav.perm == 0] <- max(pos.perm - sum(fav.perm), 0)
    }

#  if (plot) {
#    plot(stepfun(val, c(0, as.numeric(fav.perm / pos.perm) )), main
     = "Null Distribution of EIC")
#    points(val[ind.count], as.numeric(fav.perm / pos.perm)[ind.
     count], col="red")
#    points(val[ind.perm], as.numeric(fav.perm / pos.perm)[ind.perm
     ], col="green")
#    points(val[-c(ind.count, which(ind.perm))], as.numeric(fav.perm
     / pos.perm)[-c(ind.count, which(ind.perm))], col="blue")
#    legend("topleft", legend = c("exact", "permutation", "
     intrapolation"), col = c("red", "green", "blue"), pch = rep(1,3))
#  }


    return(list("val" = val, "pos.perm" = pos.perm, "fav.perm" = fav.
        perm))
}
```

A.4.3 eic_posI.cpp

```
/* These functions calculate the number of favorable permutations of
    the null distribution
```

```
 * of the eic classifier for a positive interval for a given set of
    parameters fp, fn,
 * n0, n1, c0, c1, pi0. posInt calculates the starting values and
    calls the recursive
 * function posInt_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

void posInt_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sum_posInt, int level, int start, int stop, int fp, int fn
    , int n0, int n1, double c0, double c1, double pi0) {

  int arr[5] = {level, start, stop, fp, fn}; // map key
  std::vector<int> itm(arr, arr+5);

  if (memo.find(itm) != memo.end()) { // map look up

    *sum_posInt = *sum_posInt + memo.at(itm);

  } else { // if key does not exist in map, calculate the
      corresponding value

    if (level == 1) { // end of recursion

      *sum_posInt = *sum_posInt + stop - start + 1;

    } else { // recursion step

      // calculate new recursion arguments
      int level_new = level - 1;
      double crt = (fp - level_new + 1) * c0 / c1 * n1 / n0;
      int mnpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) : ceil(
          crt);
      crt = level_new * c0 / c1 * n1 / n0;
      int mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) : ceil(
          crt);
      int start_new = mnpsr + (fp - level_new) + 1;
      int stop_new = (n1 - fn + fp) - (mxpsr + (level_new - 1));

      mpz_class bfr(*sum_posInt);

      for (int i = start; i <= stop; i = i + 1) {
          // call recursion
          posInt_rec(memo, sum_posInt, level_new, std::max(i+1, start_
              new), stop_new, fp, fn, n0, n1, c0, c1, pi0 );
        }

        mpz_class aftr(*sum_posInt);
        memo.insert( std::make_pair(itm, aftr - bfr)); // save pair of
            (key, value) into map

      }
    }
  }


// [[Rcpp::export]]
SEXP posInt(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);
  std::map <std::vector<int>, mpz_class> memo; // create map

  for (int i = 0; i < ncol; i = i + 1) {

    mpz_class sums(0);
    int fp = clst(0,i);
    int fn = clst(1,i);

    if (fp == 0) {
      sums = 1;
      perm[i] = sums;
    } else {

      // calculate initial values for recursion
      int level_init = fp;
      double crt = c0 / c1 * n1 / n0;
      int mnpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) : ceil(
          crt);
      crt = fp * c0 / c1 * n1 / n0;
      int mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) : ceil(
          crt);
      int start_init = mnpsr + 1;
      int stop_init = (n1 - fn + fp) - (mxpsr + (fp - 1));

      // start recursion
      posInt_rec(memo, &sums, level_init, start_init, stop_init,
          clst(0,i), clst(1,i), n0, n1, c0, c1, pi0);
      perm[i] = sums;
    }
  }

  memo.clear(); // delete map
  return(wrap(perm));
}
```

A.4.4 eic_negI.cpp

```
/* These functions calculate the number of favorable permutations of
     the null distribution
 * of the eic classifier for a negative interval for a given set of
     parameters fp, fn,
 * n0, n1, c0, c1, pi0. negInt calculates the starting values and
     calls the recursive
 * function negInt_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

void negInt_rec(std::map <std::vector<int>, mpz_class>& memo_negInt,
     mpz_class* sum_negInt, int level, int start, int stop, int fp,
    int fn, int n0, int n1, double c0, double c1, double pi0) {

  int arr[5] = {level, start, stop, fp, fn}; // map key
  std::vector<int> itm(arr, arr+5);

  if (memo_negInt.find(itm) != memo_negInt.end()) { // map look up

    *sum_negInt = *sum_negInt + memo_negInt.at(itm);

  } else { // if key does not exist in map, calculate the
      corresponding value

    if (level == 1) { // end of recursion

      *sum_negInt = *sum_negInt + stop - start + 1;

    } else { // recursion step

      // calculate new recursion arguments
      int level_new = level - 1;
      double crt = (fn - level_new + 1) * c1 / c0 * n0 / n1;
```

```
        int mnpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
            ceil(crt);
        crt = level_new * c1 / c0 * n0 / n1;
        int mxpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
            ceil(crt);
        int start_new = mnpsr + (fn - level_new) + 1;
        int stop_new = (n0 - fp + fn) - (mxpsr + (level_new - 1));

        mpz_class bfr(*sum_negInt);

        for (int i = start; i <= stop; i = i + 1) {
          // call recursion
          negInt_rec(memo_negInt, sum_negInt, level_new, std::max(i+1,
              start_new), stop_new, fp, fn, n0, n1, c0, c1, pi0 );
        }

        mpz_class aftr(*sum_negInt);
        memo_negInt.insert( std::make_pair(itm, aftr - bfr)); // save
            pair of (key, value) into map

      }
    }
}


// [[Rcpp::export]]
SEXP negInt(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);
  std::map <std::vector<int>, mpz_class> memo; // create map

  for (int i = 0; i < ncol; i = i + 1) {

    mpz_class sums(0);
    int fp = clst(0,i);
    int fn = clst(1,i);

    if (fn == 0) {
      sums = 1;
      perm[i] = sums;
    } else {

      // calculate initial values of recursion
      int level_init = fn;
      double crt = c1 / c0 * n0 / n1;
      int mnpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
          ceil(crt);
      crt = fn * c1 / c0 * n0 / n1;
      int mxpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
          ceil(crt);

      int start_init = mnpsr + 1;
      int stop_init = (n0 - fp + fn) - (mxpsr + (fn - 1));

      if (start_init > stop_init) {
        sums = 0;
        perm[i] = sums;
      } else {

        // start recursion
        negInt_rec(memo, &sums, level_init, start_init, stop_init,
            clst(0,i), clst(1,i), n0, n1, c0, c1, pi0);
        perm[i] = sums;

      }
    }
  }

  memo.clear(); // delete map
  return(wrap(perm));
}
```

A.4.5 eic_posC.cpp

```
/* These functions calculate the number of favorable permutations of
    the null distribution
 * of the eic classifier for a positive complement for a given set
    of parameters fp, fn,
 * n0, n1, c0, c1, pi0. posComp calculates the starting values and
    calls the recursive
 * function posComp_rec.
 */

#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

void posComp_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int tp_h, int tp_o,
    int fp_h, int fp_o, int n0, int n1, double c0, double c1, double
    pi0) {
```

```
    int arr[7] = {level, start, stop, tp_h, tp_o, fp_h, fp_o}; // map
        key
    std::vector<int> itm(arr, arr+7);

    if (memo.find(itm) != memo.end()) { // map look up

      *sums = *sums + memo.at(itm);

    } else { // if key does not exist in map, calculate the
        corresponding value

      if (level == 1) { // end of recursion

        *sums = *sums + stop - start + 1;

      } else { // recursion step

        // calculate new recursion arguments
        int level_new = level - 1;
        double crt = level_new * c0 / c1 * n1 / n0;
        int mnpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt) : ceil(
            crt);
        crt = std::max((fp_o + (fp_h - level_new + 1)) * c0 / c1 * n1
            / n0 - tp_o, 0.0);
        int mxpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt) : ceil(
            crt);
        int start_new = mnpsr + level_new;
        int stop_new = (fp_h + tp_h) - (mxpsr + (fp_h - level_new));


        mpz_class bfr(*sums);

        for (int i = start; i <= stop; i = i + 1) {
          posComp_rec(memo, sums, level_new, start_new, std::min(stop_
              new, i - 1), tp_h, tp_o, fp_h, fp_o, n0, n1, c0, c1, pi0)
              ;
        }

        mpz_class aftr(*sums);
        memo.insert( std::make_pair(itm, aftr - bfr)); // save pair of
            (key, value) into map

      }
    }
}


// [[Rcpp::export]]
SEXP posComp(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
  double c1, double pi0) {

// declaration of variables
int ncol = clst.ncol();
std::vector <mpz_class> perm(ncol);
std::map <std::vector<int>, mpz_class> memo; // create map
int fp, fn, mid;
int l_minpsr, r_minpsr, l_mxpsr, r_mxpsr;
int l_level_init, r_level_init, l_start_init, r_start_init, l_stop
   _init, r_stop_init;
double crt;
mpz_class sum_lft(0), sum_rght(0), sum_tot(0);

for (int i = 0; i < ncol; i = i + 1) {

  // setting
  fp = clst(0,i);
  fn = clst(1,i);
  sum_lft = 0;
  sum_rght = 0;
  sum_tot = 0;

  crt = static_cast<double>(n1 - fn + fp - 1) / 2;
  int mx = static_cast<int>(ceil(crt));
  mid = ((n1 - fn + fp - 1) % 2 == 0) ? -1 : mx;

  for (int l = 1; l <= mx; l = l + 1) {
    for (int fp_l = 0; fp_l <= fp; fp_l = fp_l + 1) {

      int tp_l = l - fp_l; // true negatives on the left side
      int tp_r = (n1 - fn) - tp_l; // true negatives on the right
         side
      int fp_r = fp - fp_l; // false negatives on the right side

      sum_lft = 0;
      sum_rght = 0;

      // set l_minpsr and r_minpsr
      if (fp_l == 0) { l_minpsr = -1;
      } else {
        crt = fp_l * c0 / c1 * n1 / n0;
        l_minpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) :
           ceil(crt);
      }

      if (fp_r == 0) { r_minpsr = -1;
      } else {
        crt = fp_r * c0 / c1 * n1 / n0;
        r_minpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) :
```

```
      ceil(crt);
}

if (tp_l >= l_minpsr & tp_r >= r_minpsr) {

  // permutations of the left complement
  if (fp_l == 0) { sum_lft = 1;
  } else {


    // calculate initial recursion arguments
    l_level_init = fp_l;
    crt = (fp_r + 1) * c0 / c1 * n1 / n0 - tp_r;
    l_mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) :
        ceil(crt);
    l_mxpsr = std::max(0, l_mxpsr);
    l_start_init = l_minpsr + (fp_l - 1) + 1;
    l_stop_init = (fp_l + tp_l) - l_mxpsr;

    if (l_start_init <= l_stop_init & l_stop_init <= fp_l +
        tp_l) {

      // start recursion
      posComp_rec(memo, &sum_lft, l_level_init, l_start_init
          , l_stop_init, tp_l, tp_r, fp_l, fp_r, n0, n1, c0,
          c1, pi0);

    } else { sum_lft = 0; }
  }

// permutations of the right complement

if ( fp_r == 0) { sum_rght = 1;
} else {

  // calculate initial recursion arguments
  r_level_init = fp_r;
  crt = (fp_l + 1) * c0 / c1 * n1 / n0 - tp_l;
  r_mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt) : ceil
      (crt);
  r_mxpsr = std::max(r_mxpsr, 0);
  r_start_init = r_minpsr + (fp_r - 1) + 1;
  r_stop_init = (fp_r + tp_r) - r_mxpsr;

  if (r_start_init <= r_stop_init & r_stop_init <= fp_r + tp
      _r) {

    // start recursion
    posComp_rec(memo, &sum_rght, r_level_init, r_start_init,
                r_stop_init , tp_r , tp_l , fp_r , fp_l , n0, n1, c0, c1,
                pi0);

            } else { sum_rght = 0; }

          }

          // overall number of permutations equals their product
          if (l == mid) {
            sum_tot = sum_tot + sum_lft * sum_rght;
          } else {
            sum_tot = sum_tot + 2 * sum_lft * sum_rght;
          }

          }
        }
      }

      perm[i] = sum_tot;

    }

    memo.clear(); // delete map

    return(wrap(perm));

  }
```

A.4.6 eic_negC.cpp

```
/* These functions calculate the number of favorable permutations of
     the null distribution
 * of the eic classifier for a negative complement for a given set
     of parameters fp, fn,
 * n0, n1, c0, c1, pi0. negComp calculates the starting values and
     calls the recursive
 * function negComp_rec.
 */
#include <wrap.hpp>
#include <map>
#include <gmpxx.h>
#include <Rcpp.h>
#include <math.h>

using namespace Rcpp;

void negComp_rec(std::map <std::vector<int>, mpz_class>& memo, mpz_
    class* sums, int level, int start, int stop, int tn_h, int tn_o,
    int fn_h, int fn_o, int n0, int n1, double c0, double c1, double
    pi0) {

  int arr[7] = {level, start, stop, tn_h, tn_o, fn_h, fn_o}; // map
     key
  std::vector<int> itm(arr, arr+7);

  if (memo.find(itm) != memo.end()) { // map look up

    *sums = *sums + memo.at(itm);

  } else { // if key does not exist in map, calculate the
     corresponding value

    if (level == 1) { // end of recursion

      *sums = *sums +  stop - start + 1;

    } else { // recursion step

      // calculate new recursion arguments
      int level_new = level - 1;
      double crt = level_new * c1 / c0 * n0 / n1;
      int mnpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
         ceil(crt);
      crt = std::max((fn_o + (fn_h - level_new + 1)) * c1 / c0 * n0
         / n1 - tn_o, -0.0001);
      int mxpsr = fabs(round(crt) - crt) < 1e-6 ? round(crt + 1) :
         ceil(crt);
      int start_new = mnpsr + level_new;
      int stop_new = (fn_h + tn_h) - (mxpsr + (fn_h - level_new));

      mpz_class bfr(*sums);

      for (int i = start; i <= stop; i = i + 1) {
        // call recursion
        negComp_rec(memo, sums, level_new, start_new, std::min(stop_
           new, i - 1), tn_h, tn_o, fn_h, fn_o, n0, n1, c0, c1, pi0)
           ;
      }

      mpz_class aftr(*sums);
      memo.insert( std::make_pair(itm, aftr - bfr)); // save pair of
          (key, value) into map

    }
  }
}
```

```
// [[Rcpp::export]]
SEXP negComp(Rcpp::NumericMatrix clst, int n0, int n1, double c0,
    double c1, double pi0) {

  // declaration of variables
  int ncol = clst.ncol();
  std::vector <mpz_class> perm(ncol);
  std::map <std::vector<int>, mpz_class> memo; // create map
  int fp, fn, mid;
  int l_minpsr, r_minpsr, l_mxpsr, r_mxpsr;
  int l_level_init, r_level_init, l_start_init, r_start_init, l_stop
     _init, r_stop_init;
  double crt;
  mpz_class sum_lft(0), sum_rght(0), sum_tot(0);

  for (int i = 0; i < ncol; i = i + 1) {

    // setting
    fp = clst(0,i);
    fn = clst(1,i);
    sum_lft = 0;
    sum_rght = 0;
    sum_tot = 0;

    crt = static_cast<double>((n0 - fp + fn)/2);
    int mx = static_cast<int>(ceil(crt));
    mid = ((n0 - fp + fn) % 2 == 0) ? mx : -1;

    for (int l = 0; l <= mx; l = l + 1) {
      for (int fn_l = 0; fn_l <= fn; fn_l = fn_l + 1) {

        int tn_l = l - fn_l;  // true negatives on the left side
        int tn_r = (n0 - fp) - tn_l; // true negatives on the right
           side
        int fn_r = fn - fn_l; // false negatives on the right side

        sum_lft = 0;
        sum_rght = 0;

        // set l_minpsr and r_minpsr
        if (fn_l == 0) { l_minpsr = -1;
        } else {
          crt = fn_l * c1 / c0 * n0 / n1;
          l_minpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt + 1) :
               ceil(crt);
        }

        if (fn_r == 0) { r_minpsr = -1;
        } else {
```

```
    crt = fn_r * c1 / c0 * n0 / n1;
    r_minpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt + 1) :
        ceil(crt);
  }

  if (tn_l >= l_minpsr & tn_r >= r_minpsr) {

    // permutations of the left complement
    if (fn_l == 0) { sum_lft = 1;
    } else {

      // calculate initial recursion arguments
      l_level_init = fn_l;
      crt = (fn_r + 1) * c1 / c0 * n0 / n1 - tn_r;
      l_mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt + 1)
          : ceil(crt);
      l_mxpsr = std::max(0, l_mxpsr);
      l_start_init = l_minpsr + (fn_l - 1) + 1;
      l_stop_init = (fn_l + tn_l) - l_mxpsr;

      if (l_start_init <= l_stop_init & l_stop_init <= fn_l +
          tn_l) {

        // start recursion
        negComp_rec(memo, &sum_lft, l_level_init, l_start_init
            , l_stop_init, tn_l, tn_r, fn_l, fn_r, n0, n1, c0,
            c1, pi0);

      } else { sum_lft = 0; }
    }

    // permutations of the right complement

    if (fn_r == 0) { sum_rght = 1;
    } else {

      // calculate initial recursion arguments
      r_level_init = fn_r;
      crt = (fn_l + 1) * c1 / c0 * n0 / n1 - tn_l;
      r_mxpsr = fabs(ceil(crt) - crt) < 1e-6 ? round(crt + 1)
          : ceil(crt);
      r_mxpsr = std::max(0, r_mxpsr);
      r_start_init = r_minpsr + (fn_r - 1) + 1;
      r_stop_init = (fn_r + tn_r) - r_mxpsr;

      if (r_start_init <= r_stop_init & r_stop_init <= fn_r +
          tn_r) {

        // start recursion
```

```
              negComp_rec(memo, &sum_rght, r_level_init, r_start_
                  init, r_stop_init, tn_r, tn_l, fn_r, fn_l, n0, n1,
                  c0, c1, pi0);

            } else { sum_rght = 0; }

          }

          // overall number of permutations
          if (l == mid) {
            sum_tot = sum_tot + sum_lft * sum_rght;
          } else {
            sum_tot = sum_tot + 2 * sum_lft * sum_rght;
          }

        }
      }
    }

    perm[i] = sum_tot;

  }

  memo.clear(); // delete map

  return(wrap(perm));

}
```

List of Figures

List of Tables

Bibliography

N. M. Adams and D. J. Hand. Comparing classifiers when the misallocation costs are uncertain. *Pattern Recognition*, 32(7):1139–1147, 1999.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practival and powerful approach to multiple testing. *Journal of the Royal Statistical Society B 57, No. 1*, 57(1):289–300, 1995.

P. J. Bickel and K. A. Doksum. *Mathematical statistics: basic ideas and selected topics, volume I*, volume 117. CRC Press, 2015.

A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recogn.*, 30(7):1145–1159, July 1997. ISSN 0031-3203. doi: 10.1016/S0031-3203(96)00142-2. URL `http://dx.doi.org/10.1016/S0031-3203(96)00142-2`.

M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis 1*, pages 131–156, 1997.

A. Delaigle, P. Hall, and J. Jin. Robustness and accuracy of methods for high dimensional data analysis based on student's t-statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):283–301, 2011. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2010.00761.x. URL `http://dx.doi.org/10.1111/j.1467-9868.2010.00761.x`.

C. Drummond and R. C. Holte. Cost curves: An improved method for visualizing classifier performance. *Machine Learning*, 65(1):95–130, Oct 2006. ISSN 1573-0565. doi: 10.1007/s10994-006-8199-5. URL `https://doi.org/10.1007/s10994-006-8199-5`.

R. O. Duda, P. E. Hart, and D. G. Stork. Pattern classification. *Intelligent Data Analysis 1 (1-4)*, pages 131–156, 2012.

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL `http://www.jstatsoft.org/v40/i08/`.

E. S. Edgington. *Randomization Tests*, pages 1182–1183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_56. URL http://dx.doi.org/10.1007/978-3-642-04898-2_56.

E. Giné, F. Götze, and D. M. Mason. When is the Student *t*-statistic asymptotically standard normal? *Ann. Probab.*, 25(3):1514–1531, 07 1997. doi: 10.1214/aop/1024404523. URL http://dx.doi.org/10.1214/aop/1024404523.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, Oct. 1999.

T. Granlund et al. GMP: the GNU multiple precision arithmetic library. https://gmplib.org, 1993–2017.

B. D. W. Group. Biomarkers and surrogate endpoints: Preferred definitions and conceptual framework. *Clinical Pharmacology and Therapeutics*, 69(3):89–95, 2001. ISSN 1532-6535. doi: 10.1067/mcp.2001.113989. URL http://dx.doi.org/10.1067/mcp.2001.113989.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, Mar. 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944968.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer, New York, NY, USA, 2001.

G. A. Heap, G. Trynka, R. C. Jansen, M. Bruinenberg, M. A. Swertz, L. C. Dinesen, K. A. Hunt, C. Wijmenga, L. Franke, et al. Complex nature of snp genotype effects on gene expression in primary human leucocytes. *BMC Medical Genomics*, 2(1):1, 2009.

J. Hernández-Orallo, P. Flach, and C. Ferri. A unified view of performance metrics: Translating threshold choice into expected classification loss. *J. Mach. Learn. Res.*, 13 (1):2813–2869, Oct. 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2503308.2503332.

J. Holland. *The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance.* MIT Press, Cambridge, MA, 1992.

J. Hsu. *Multiple Comparisons: Theory and Methods.* Guilford School Practitioner. Taylor & Francis, Adingdon, UK, 1996. ISBN 9780412982811. URL https://books.google.at/books?id=8AK8PUbw3lsC.

J. M. Hyman. Accurate monotonicity preserving cubic interpolation. *SIAM Journal on Scientific and Statistical Computing*, 4(4):645–654, 1983.

J. Jaeger, R. Sengupta, and W. Ruzzo. *Improved gene selection for classification of microarrays.*, pages 53–64. 2003.

J. Kittler. Feature set search algorithms. *Pattern recognition and signal processing*, 1978. URL `http://ci.nii.ac.jp/naid/80014031027/en/`.

T. A. Knijnenburg, L. F. A. Wessels, M. J. T. Reinders, and I. Shmulevich. Fewer permutations, more accurate p-values. *Bioinformatics*, 25(12):i161–i168, June 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp211. URL `http://dx.doi.org/10.1093/bioinformatics/btp211`.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(97)00043-X. URL `http://www.sciencedirect.com/science/article/pii/S000437029700043X`.

I. Levner. Feature selection and nearest centroid classification for protein mass spectrometry. *BMC Bioinformatics*, 6(1):68, Mar 2005. ISSN 1471-2105. doi: 10.1186/1471-2105-6-68. URL `https://doi.org/10.1186/1471-2105-6-68`.

T. J. MacDonald, K. M. Brown, B. LaFleur, K. Peterson, C. Lawlor, Y. Chen, R. J. Packer, P. Cogen, and D. A. Stephan. Expression profiling of medulloblastoma: PDGFRA and the RAS/MAPK pathway as therapeutic targets for metastatic disease. *Nature Genetics*, 29(2):143–152, 2001.

R. Miller. *Simultaneous Statistical Inference.* Springer Series in Statistics. Springer-Verlag, Heidelberg, Deutschland, 1981. ISBN 9780387905488. URL `https://books.google.at/books?id=kPQhL9rofgYC`.

M. Mura, M. Anraku, Z. Yun, K. McRae, M. Liu, T. K. Waddell, L. G. Singer, J. T. Granton, S. Keshavjee, and M. de Perrot. Gene expression profiling in the lungs of patients with pulmonary hypertension associated with pulmonary fibrosis. *CHEST Journal*, 141(3):661–673, 2012.

J. Neyman. Sur un teorema concernente le cosidette statistiche sufficienti. *Ist. Ital. Att.*, 6:320–334, 1935.

H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria on fa max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1226–1238, 2005.

I. Pigeot. Basic concepts of multiple tests — a survey. *Statistical Papers*, 41(1):3–36, Jan 2000. ISSN 1613-9798. doi: 10.1007/BF02925674. URL `https://doi.org/10.1007/BF02925674`.

F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning 42 (3)*, pages 203–231, 2001.

F. J. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 445–453, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8. URL http://dl.acm.org/citation.cfm?id=645527.657469.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL https://www.R-project.org/.

J. J. Schlesselman and P. D. Stolley. *Case-Control Studies : Design, Conduct, Analysis*. Oxford University Press, Oxford, UK, 1982.

G. K. Smyth et al. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat Appl Genet Mol Biol*, 3(1):3, 2004.

Student. The probable error of a mean. *Biometrika*, 6(1):1–25, 1908.

Y. Su, T. Murali, V. Pavlovic, M. Schaffer, and S. Kasif. Rankgene: identification of diagnostic genes based on expression data. *Bioinformatics*, 19(12):1578–1579, 2003. doi: 10.1093/bioinformatics/btg179. URL +http://dx.doi.org/10.1093/bioinformatics/btg179.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

S. Tsukamoto, T. Ishikawa, S. Iida, M. Ishiguro, K. Mogushi, H. Mizushima, H. Uetake, H. Tanaka, and K. Sugihara. Clinical significance of osteoprotegerin expression in human colorectal cancer. *Clinical Cancer Research*, 17(8):2444–2450, 2011.

J. Tukey. *The problem of multiple comparisons*, volume In The Collected Works of John W. Tukey VIII. Multiple Comparisons. Chapman and Hall, New York, 1953.

S. Viaene and G. Dedene. Cost-sensitive learning and decision making revisited. *European Journal of Operational Research 166*, 166:212–220, 2005.

B. L. Welch. The generalization of student's problem when several different population variances are involved. *Biometrika 34 (1–2)*, 34(1/2):28–35, 1947.

F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6): 80–83, Dec. 1945. ISSN 00994987. doi: 10.2307/3001968. URL http://dx.doi.org/10.2307/3001968.

A. Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 987–996, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics. URL http://dl.acm.org/citation.cfm?id=1496770.1496877.

B. Wu, T. Abbott, D. Fishman, W. McMurray, G. Mor, K. Stone, D. Ward, K. Williams, and H. Zhao. Comparison of statistical methods for classification of ovarian cancer using mass spectrometry data. *Bioinformatics*, 19(13):1636–1643, 2003. doi: 10.1093/bioinformatics/btg210. URL +http://dx.doi.org/10.1093/bioinformatics/btg210.

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2005.00503.x. URL http://dx.doi.org/10.1111/j.1467-9868.2005.00503.x.

Curriculum Vitae

Personal

Name:	Fabian Schroeder
Date of birth:	November 5th, 1983
Place of birth:	Vienna, Austria
Nationality:	Austrian
Children:	Sophia (05/2014), Valerie (10/2015), Moritz (07/2017)

Education

since 11/2014	Ph.D. in Statistics, Vienna University of Technology, Austria
2009 – 2012	M.Sc. in Statistcs, Vienna University of Technology, Austria
2006 – 2009	B.Sc. in Statistics and Mathematics for Economics, Vienna University of Technology, Austria
2003 – 2009	M.Sc. in Economics, University of Vienna
1993 – 2001	Gymnasium BGXIX, Vienna, Austria

Working Experience

since 01/2018	Project Assistant, Vienna University of Technology
11/2013 – 12/2017	Ph.D. Student, Austrian Institute of Technology, Vienna
03/2012 – 12/2012	Intern, Austrian Institute of Technology, Vienna
05/2003 – 06/2012	Research and Personal Assistant to Prof. John Naisbitt Megatrends Ltd., Vienna
02/2002 – 05/2003	Gedenkdiener, CJVMA, Montreal and Yad Vashem, Jerusalem