Identifying and Characterising Anomalies in Data

Proefschrift

voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica
aan de Universiteit Antwerpen
te verdedigen door

Koen SMETS

Promotor: prof. dr. Bart Goethals

Antwerpen, 2012

Identifying and Characterising Anomalies in Data
Nederlandse titel: *Identificeren en karakteriseren van onregelmatigheden in gegevens*

The research in this thesis was supported by an umbrella grant of the University of Antwerp for FWO candidates and a Ph.D. Fellowship of the Research Foundation – Flanders (FWO).

Through every rift of discovery some seeming anomaly drops out of the darkness, and falls, as a golden link into the great chain of order.

— Edwin Hubbell Chapin

Acknowledgements

Looking back, on the bumpy road towards this dissertation, a mixture of feelings, memories and thoughts overwhelm me when considering my life as a doctoral student. I really feel privileged for having been able to bang my head against several brick walls, to discover valuable knowledge along the way, and to be surrounded by so many strong people who all wanted, always kept believing, and made sure that I would reach this milestone successfully.

First, I would like to thank my former supervisor Brigitte Verdonk, not only for giving me the opportunity and the freedom to explore several roads before ending under Bart's wings, but also for pushing me to make sure that every U-turn I took resulted in tangible outcomes.

Next, I would like to thank my supervisor Bart Goethals, for dropping by my office one day making me realise that I had been completely blind-sighted for the data mining part in ADReM. I am ultimately grateful for *all* following visits which influenced me in no small way. I can only be proud that the latest publication rounds up the research we initiated after my first skeptical steps in mining itemsets.

For this, and other insights during inspiring discussions, I would like to thank my co-author and loudest (room)mate Jilles Vreeken. Along with Bart you definitely helped me connecting the dots.

Naturally, I would like to thank the other members of my doctoral jury: Arno Siebes, for indirectly bringing compression and MDL to my attention; Thomas Gärtner for reassuring me that the step from kernels to itemsets is not that far fetched; Floris Geerts for reminding me that anomalies play an essential role in cleaning data and Jan Paredaens for fulfilling the role of chairman.

Furthermore, I would like to thank all other people that have crossed my path the past years for putting my mind either at work or at ease.

Last but not the least, I would like to take the opportunity to express my ultimate gratitude towards my girlfriend Karo, my sister Nele, my parents Mieke and Guy, my parents-in-law Erna and Gilbert, and my closest friends Przemek and Timmy for their love and support.

Thanks!

Koen Smets
Antwerpen, 2012

Contents

Publications

- **K. Smets**, B. Verdonk, and E. M. Jordaan. Evaluation of performance measures for SVR hyperparameter selection. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN), Orlando, FL*, pages 637–642, 2007.

- **K. Smets**, B. Goethals, and B. Verdonk. Automatic vandalism detection in Wikipedia: Towards a machine learning approach. In *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy (WikiAI), Chicago, IL*, pages 43–48, 2008.

- **K. Smets**, B. Verdonk, and E. M. Jordaan. Discovering novelty in spatio/temporal data using one-class support vector machines. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN), Atlanta, GA*, pages 2956–2963, 2009.

- **K. Smets** and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 804–815, 2011.

- N. Remmerie, T. D. Vijlder, D. Valkenborg, K. Laukens, **K. Smets**, J. Vreeken, I. Mertens, S. C. Carpentier, B. Panis, G. D. Jaeger, R. Blust, E. Prinsen, and E. Witters. Unraveling tobacco BY-2 protein complexes with BN PAGE/LC–MS/MS and clustering methods. *Journal of Proteomics*, 74(8):1201 – 1217, 2011.

- T. Vu, D. Valkenborg, **K. Smets**, K. Verwaest, R. Dommisse, F. Lemiere, A. Verschoren, B. Goethals, and K. Laukens. An integrated workflow for robust alignment and simplified quantitative analysis of NMR spectrometry data. *BMC Bioinformatics*, 12(1):405, 2011.

- **K. Smets** and J. Vreeken. SLIM: Directly mining descriptive patterns. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA*, 2012.

CHAPTER 1

Introduction

Data mining is a step in the knowledge discovery in databases (KDD) process that consists in applying data analysis and discovery algorithms with the goal to find unsuspected relationships and to summarise the data in ways that are both understandable and useful [25]. These relationships and summaries are often referred to as patterns and models.

When modeling data, one differentiates typically between predictive and descriptive modeling. A predictive model is built, or learnt, from the data and is typically meant to be used to predict the value of an unknown output variable depending on one or more known input variables. Depending on the type of the response variable, e.g. categorical or quantitative, it is respectively called classification or regression. The goal of descriptive modeling, on the other hand, is to provide insightful descriptions of the data, e.g. by grouping the data into clusters. Other data mining algorithms are focused on the discovery of patterns and rules. For example, the task of finding combinations of items that occur frequently in databases.

Besides discovering these regularities, detecting exceptions, or anomalies, in data can be equally interesting. For example, anomalous traffic in a computer network could signal the failure of one or more devices or that a hacker has penetrated the network, while anomalies in credit card transfers could indicate credit card or identity theft. In healthcare, anomalous sensor readings when monitoring patients at the intensive care unit could signify organ failure, or when screening newborns abnormal values in a blood sample could indicate the presence of a rare disease.

In general, anomalies in data, often also referred to as outliers, are observations that deviate from the expected normal behaviour [14]. Two problems are commonly associated with anomalies: anomaly detection and anomaly characterisation. Anomaly detection refers to the problem of finding anomalies in data, while anomaly characterisation provides insight for the anomalousness.

An abstract approach to detect anomalies in data is to model normal behaviour and declare any observation that differs strongly from this normal model as an anomaly. Modeling the norm of the data, however, is not a trivial task and therefore in its general form, this problem is not easy to solve. Over time, a variety of anomaly detection techniques have been developed adopting concepts from diverse research communities such as statistics, machine learning, data mining and information theory, solving specific instantiations of the problem that depends among others on the type of input data, the availability of labeled training data, ... These specifications differ from application to application.

Example of applications where anomaly detection is typically used and which serve as main motivation for the research in this thesis are intrusion and fraud detection, monitoring in health care and industry, and screening for rare diseases. These applications have in common that there is an abundance of examples for the normal cases, but none, or only a handful, for the anomalies. This is due to the fact that it is either very expensive, dangerous, or virtually impossible to acquire (many) labeled examples for abnormal situations. Standard binary (or multi-class) classification techniques need enough labeled training data for each class to build an accurate predictive model and thus cannot be applied directly. Since we are building a model for the normal behaviour given only training data from one class, this problem setting is known as one-class classification.

Anomalies are related to noise and novelties in the data [49]. All these terms refer to objects or patterns in data that are typically unexpected. Adopting the definitions by Chandola et al. [14], the main difference between noise and anomalies lies in the interest of the data analyst. Anomalies are considered meaningful to a data analyst, while noise hinders modeling and obfuscates true patterns in the data. Novelties, on the other hand, are both unexpected and interesting to a data analyst. Anomalies should however be treated as abnormalities, while novelties are typically incorporated into the normal model after being detected. Since techniques to identify anomalies, or outliers, are often used to detect noise or novelties and vice versa many authors, including myself, do not always follow these strict definitions.

Identification of anomalies alone is not enough however; characterisation of anomalies is also very important. This goes in general for all kinds of patterns in data, but since anomalies translate to significant, and often critical, actionable information, descriptions are especially important. For example, a human operator will not likely shut down a chemical installation based on detection of a few anomalies if there is no good explanation to do so. Similarly, medical doctors are ultimately responsible for their patients, and hence will not trust a black-box technique telling them a patient has a rare disease if it cannot explain why this must be so. Likewise, Wikipedians will not revert an edit if no clear evidence is presented that a page has been vandalised and thus need a good explanation and characterisation.

In this dissertation, we aim developing efficient data mining techniques to build, starting from the available data with limited human effort, models that accurately identify anomalies in (new) data and characterise these in an understandable way.

1.1 Thesis Outline

The outline of the remainder of this thesis is as follows.

- **Chapter 2** investigates how to detect abnormal or novel events embedded in spatio/temporal data. To this end, we present an algorithm based on Support Vector Data Description (SVDD). The algorithm uses an extended representation of the spatio/temporal data, a tensor product kernel to separately deal with the distinct features of time and measurements, and a voting function which identifies anomalies based on different representations of the time series in a robust way.
- **Chapter 3** explores to which extend straightforward data-driven techniques can be employed to distinguish vandalism from legitimate edits on Wikipedia.
- **Chapter 4** shows how to identify and characterise anomalies in binary or transaction data. We use the Minimal Description Length (MDL) principle to model the normal behaviour and detect anomalies as deviation from the expected length in bits of the compressed instances. By analysing the itemsets, the small blocks used to build the model and to compress instances, we can characterise the decisions, and explain what changes would lead to a different verdict. We also give a method that, given a few anomalous examples, estimates the distribution of encoded lengths for the anomalies.
- **Chapter 5** addresses the issues, manifested in dense and large datasets, of the underlying algorithm that is used to build the models for the normal behaviour in Chapter 4. We therefore introduce an any-time, one-phase alternative for mining high-quality data descriptions directly and efficiently from transaction data.
- **Chapter 6** summarises the main contributions of the dissertation and rounds up with concluding remarks.

CHAPTER 2

Identifying Anomalies in Spatio/Temporal Data

Anomaly or novelty detection in spatio/temporal data refers to the automatic identification of abnormal or novel events embedded in data that occur at a specific location/time. Traditional techniques used in process control to identify anomalies are not robust for noise in the data set.

We present an algorithm based on the support vector machine approach for data description. This technique is intrinsically robust for anomalies in the data set, but to make it work several extensions are needed that form the main contribution of this work: an extended representation of the spatio/temporal data, a tensor product kernel to separately deal with the distinct features of time and measurements, and a voting function which identifies anomalies based on different representations of the time series in a robust way.

Experimental results on both artificial and real data demonstrate that our algorithm performs significantly better than other standard techniques used in process control.

This chapter is based on work published as [72]:
K. Smets, B. Verdonk, and E. M. Jordaan. Discovering novelty in spatio/temporal data using one-class support vector machines. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN), Atlanta, GA*, pages 2956–2963, 2009.

2.1 Introduction

In many real-life applications, information gathered from measurements is essential to ensure the quality of products and to enable control of a production process. Advances in data capture technology and the availability of inexpensive storage result in huge amounts of measurements. This makes it harder for human experts to analyse the data in a timely and cost effective way. Therefore, algorithms to detect malfunctioning or abnormal behaviour, and which automate the data analysis, are needed to make the construction of monitoring systems more efficient.

Automated data analysis for the detection of anomalies is not a new concept. For years data-driven non-parametric computational intelligence techniques have been used for flight control systems [43], to monitor power plants [94], ... Anomaly detection (outlier identification) is a standard problem in classical statistics [3] as well as in data mining [75] and machine learning [45, 50]. A state-of-the-art contribution to the toolbox of non-parametric anomaly detection methods is Support Vector Data Description (SVDD) [66, 79]. This approach is based on the support vector machine (SVM) algorithm, that has popularised and stimulated research in kernel-based learning methods [67].

In the last few years, the SVDD (or one-class SVM) method has been applied for various tasks [21, 27, 44]. Most authors concentrate on detecting anomalies in static data, where they use the data as-is, as in the analysis of mass spectral data [80], to detect whether the object under inspection is abnormal when considered as a whole. Little attention is given to apply the method, or a closely related online variant, for detecting anomalies or novelties in time-series [17, 46].

The present work is motivated by anomaly detection problems in production processes, like the detection of abnormal events during the production of chemical products. These processes are monitored by inspecting various input/output relations that change over time. Figure 2.1 shows data from a typical chemical batch process, where 6 variables are monitored from start until finish. The aim is to detect anomalies like abnormal peaks, time shifts and phases that take too long. Therefore our goal is different from the goal in time series analysis, where the emphasis is on identifying patterns and/or on forecasting.

The problem of determining whether or not a particular batch is typical or not, is one that has been studied extensively in the field of process chemometrics [20]. The most widely used techniques are principal component analysis (PCA) and partial least squares (PLS). In order to make use of these techniques, the data needs to be unfolded [92]. The batch-level unfolding PCA is more sensitive to the overall batch variation while the observation level unfolding PLS is more sensitive to localised batch variation. Another drawback of both of these techniques is that the data from the different batches are required to be of the same length, i.e. of the same duration. In reality this, of course, is seldom the case. These techniques have been used to detect abnormal batches based on some correlation with a quality variable, but only after a batch was completed. Therefore, these techniques are not suitable to detect within batch anomalies in the spatio/temporal data setting.

In order to detect anomalies inside a particular batch, it will not suffice to construct a global model on the whole time interval and compare the batch as a whole to this model. This could only respond to queries whether or not to consider the whole batch as anomalous or not, but would not give us any clue where the batch

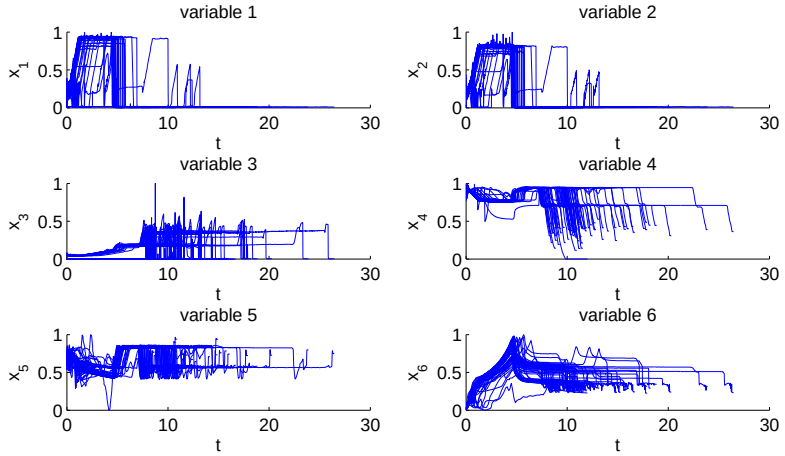


Figure 2.1: Real data coming from the production process of a chemical product, denoted as 'ProdX' for confidentiality reasons, at Dow Benelux. The data set consists of 110 batches. In each batch 6 variables x_i are monitored over time t.

has gone wrong. To accomplish the latter, we need to deduce for every sample point in time whether it is still on track or not.

Traditional techniques used in process control to achieve this are based on the computation of the point-wise mean and standard deviation over all training batches, or on the point-wise minimum and maximum value of all the samples [59]. An obvious drawback of these techniques is sensitivity to peculiarities in the training set. Since the technique we propose is based on the SVDD approach, it is intrinsically robust for noise and outliers in the data set.

To make the technique based on the SVM approach work, several extensions are needed which form the contributions of this chapter. Firstly, we present an extended data representation to reflect the vicinity of data within and across batches. Secondly, we use a composite tensor product kernel to relate the time and the data part of the representation respectively. Thirdly, we introduce a robust identification function which combines the results of several SVDD models built from different, but closely related, representations of the data.

The rest of the chapter is organised as follows. We start in Section 2.2 with a brief introduction to Support Vector Data Description, after which we give a review of the theory behind composite kernels. In Section 2.3 we present our algorithm for detecting anomalies in spatio/temporal data. The settings of the computational experiments are described in Section 2.4 and are analysed in Section 2.5. In Section 2.6, we round up with discussion and we conclude in Section 2.7.

2.2 Preliminaries

This section first briefly reviews the Support Vector Data Description (SVDD), for more details we refer to [79]. Thereafter we present the tensor product composite kernel that will be used to represent and to incorporate space/time similarities.

Support Vector Data Description

In general terms, the idea behind the SVDD method is to construct a close boundary around the training data, representating the normal scenario. In order to do so, one searches for a hypersphere that describes the data as well as possible, i.e. the hypersphere with the smallest volume. Like in the standard SVM algorithm for binary classification we allow some misclassification, i.e. we tolerate that some objects lie outside the hypersphere. This approach makes the algorithms more robust for potential anomalies present in the training data.

Formalising the above, in SVDD [79] we are given a set of unlabeled vectors $\{\boldsymbol{x}_i,\ i = 1 \dots m\}$, where $\boldsymbol{x}_i \in \mathscr{X}$. Here we assume for simplicity that the input space $\mathscr{X} \subseteq \mathbb{R}^n$. In general, a nonlinear function $\Phi(\boldsymbol{x})$ maps an object $\boldsymbol{x}$ from the input space $\mathscr{X}$ into a large, or even infinite, dimensional feature space $\mathscr{H}$. Constructing the hypersphere requires solving the following convex quadratic optimisation problem

$$\begin{aligned} \underset{R\in\mathbb{R},\boldsymbol{a}\in\mathscr{H},\boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimize}} \quad & F(R,\boldsymbol{a},\boldsymbol{\xi}) = R^2 + \frac{1}{\nu m}\sum_{i=1}^{m}\xi_i \\ \text{subject to } & \|\Phi(\boldsymbol{x}_i) - \boldsymbol{a}\|^2 \le R^2 + \xi_i \text{ and } \xi_i \ge 0 \end{aligned} \tag{2.1}$$

where $\boldsymbol{a}$ is the center of the hypersphere in feature space, R is the radius of the sphere and $\nu \in (0,1)$ a parameter to trade-off the radius of the hypersphere and the number of vectors falling outside the constructed sphere. The decision function is readily available as

$$f(\boldsymbol{z}) = \operatorname{sgn}(R^2 - \|\Phi(\boldsymbol{z}) - \boldsymbol{a}\|^2) \tag{2.2}$$

and detects whether a newly presented object $\Phi(\boldsymbol{z})$ lies inside or outside the hypersphere in feature space.

After introducing Lagrange multipliers α_i for each vector $\Phi(\boldsymbol{x}_i)$, the dual problem of the optimisation problem (2.1) can be written as

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i\langle\Phi(\boldsymbol{x}_i),\Phi(\boldsymbol{x}_i)\rangle \\ & \qquad - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\langle\Phi(\boldsymbol{x}_i),\Phi(\boldsymbol{x}_j)\rangle \\ \text{subject to } & \sum_{i=1}^{m}\alpha_i = 1 \text{ and } 0 \le \alpha_i \le \frac{1}{\nu m} \end{aligned} \tag{2.3}$$

Solving the dual problem leads to

$$\boldsymbol{a} = \sum_{i=1}^{m}\alpha_i\Phi(\boldsymbol{x}_i).$$

As distances are closely related to dot products, we can also rewrite the decision function only in terms of dot products

$$f(\boldsymbol{z}) = \operatorname{sgn}\Big(R^2 - \sum_{i,j} \alpha_i \alpha_j \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)\rangle + 2\sum_i \alpha_i \langle \Phi(\boldsymbol{z}), \Phi(\boldsymbol{x}_i)\rangle - \langle \Phi(\boldsymbol{z}), \Phi(\boldsymbol{z})\rangle\Big). \tag{2.4}$$

The famous kernel trick is the procedure of using a kernel function in input space $\mathscr{X}$ to replace the inner product of two vectors in feature space $\mathscr{H}$ [67]. Accordingly, the hypersphere in feature space $\mathscr{H}$ becomes a decision boundary in the input space $\mathscr{X}$. In order to do so, we replace the occurrences of the dot products in (2.3) and (2.4) by

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z})\rangle.$$

There are many admissible choices for the kernel function. The most widely used in SVDD is the Gaussian RBF kernel, i.e.

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{z}\|^2}{2\sigma^2}\right).$$

When $k(\boldsymbol{x}, \boldsymbol{x})$ evaluates to a constant, as is the case for the Gaussian RBF kernel, one can show that SVDD is equivalent to one-class SVM [79]. Instead of fitting a soft hypersphere around the training data, one-class SVM constructs a hyperplane in feature space to separate as many mapped vectors as possible from the origin [66].

According to (2.4), any vector $\boldsymbol{z}$ with $f(\boldsymbol{z}) = -1$ is an anomaly. Moreover, it can be proved that a vector $\boldsymbol{x}_i$ in the training set is an anomaly if and only if its corresponding Lagrange multiplier α_i is $\frac{1}{vm}$. The other support vectors $0 < \alpha_i < \frac{1}{vm}$ lie on the hypersphere in feature space. We refer to the former as non-bound support vectors and to the latter as bound support vectors. For both one-class SVM and SVDD we know that v is an upper bound for the fraction of anomalies over all training samples and a lower bound for the fraction of support vectors [66].

The main motivating, and commonly accepted, reasons to choose SVDD as our anomaly detection algorithm are the following. Firstly, it uses simple geometrical concepts to construct the decision boundary. Secondly, by using regularisation, the algorithm is intrinsically robust for anomalies present in the data set. Thirdly, because it is a kernel-based method that only relies on in-products or closely related concepts, like distances, we can plug in kernels so that the data description is able to construct a tighter decision boundary. As we will see later, the use of kernels also allows us to exploit time/space similarities. Finally, a more general version of SVDD is capable of including domain knowledge in the form of negative samples [79].

Tensor Product Composite Kernel

In [47], Mak et al. introduce the general concept of composite kernels together with two different forms, namely the direct sum kernel and the tensor product kernel. Composite kernels can be useful to combine various attributes of the data, each with different characteristics, while preserving for each attribute a distinct representation in feature space. As will be explained in Section 2.3, we make use of the tensor

product composite kernel to separately deal with the distinct features of the time and the measurements part of the data respectively.

For the sake of completeness, we repeat the definition of Mak et al. [47] and add a proof that the tensor product kernel is a valid kernel.

Definition 1 (Composite Kernel)**.** Consider vectors $\boldsymbol{x}_i$ composed as

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{x_{1}}_i & \boldsymbol{x_{2}}_i & \dots & \boldsymbol{x_{R}}_i \end{bmatrix}$$

and a mapping for each constituent $\boldsymbol{x_r}_i$, $r = 1, \dots, R$, via a separate kernel $k_r(\cdot,\cdot)$, to $\Phi_r(\boldsymbol{x_r}_i)$, and construct $\Phi(\boldsymbol{x}_i)$ as

$$\Phi(\boldsymbol{x}_i) = \begin{bmatrix} \Phi_1(\boldsymbol{x_1}_i) & \Phi_2(\boldsymbol{x_2}_i) & \dots & \Phi_R(\boldsymbol{x_R}_i) \end{bmatrix},$$

then the similarity between two vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the composite kernel-induced feature space $\mathscr{H}$ is measured by

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = G\Big(k_1(\boldsymbol{x_1}_i, \boldsymbol{x_1}_j), \dots, k_R(\boldsymbol{x_R}_i, \boldsymbol{x_R}_j)\Big)$$

where G is some function that combines the constituent kernels $k_r(\cdot,\cdot)$, $r = 1, \dots, R$ into a valid composite kernel $k(\cdot,\cdot)$.

Proposition 1 (Tensor/Hadamard/Schur Product Kernel)**.** *If $k_r(\cdot,\cdot)$, $r = 1, \dots, R$, are valid kernels, so is*

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{r=1}^{R} k_r(\boldsymbol{x_r}_i, \boldsymbol{x_r}_j).$$

We generalise the proof found in [32], which shows that the "Hadamard product in kernel space becomes the tensor product in feature space" for $R = 2$.

Proof. The new feature vector is defined as a rank R tensor (i.e. with R indices): one index for each of the original feature vectors.

$$\Phi(\boldsymbol{x_i})_{I_1 I_2 \dots I_R} = \prod_{r=1}^{R} \Phi_r(\boldsymbol{x_r}_i)_{I_r}$$

means that the $(I_1, I_2, \dots, I_R)$ component of the new feature vector is the product of the I_1^{th} component of $\Phi_1(\boldsymbol{x_1}_i)$, the I_2^{th} component of $\Phi_2(\boldsymbol{x_2}_i)$ … and the I_R^{th} component of $\Phi_R(\boldsymbol{x_R}_i)$.

This being said, we have the following derivation, using the natural component-wise inner product for $\Phi(\boldsymbol{x}_i)$:

$$\begin{aligned} k(\boldsymbol{x}_i,\boldsymbol{x}_j) &= \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j)\rangle \\ &= \sum_{I_1,I_2,\ldots,I_R} \Phi(\boldsymbol{x}_i)_{I_1 I_2 \ldots I_R} \Phi(\boldsymbol{x}_j)_{I_1 I_2 \ldots I_R} \\ &= \sum_{I_1,I_2,\ldots,I_R} \prod_{r=1}^{R} \Phi_r(\boldsymbol{x_r}_i)_{I_r} \prod_{r=1}^{R} \Phi_r(\boldsymbol{x_r}_j)_{I_r} \\ &= \prod_{r=1}^{R} \sum_{I_r} \Phi_r(\boldsymbol{x_r}_i)_{I_r} \Phi_r(\boldsymbol{x_r}_j)_{I_r} \\ &= \prod_{r=1}^{R} \langle \Phi_r(\boldsymbol{x_r}_i), \Phi_r(\boldsymbol{x_r}_j)\rangle \\ &= \prod_{r=1}^{R} k_r(\boldsymbol{x_r}_i, \boldsymbol{x_r}_j). \end{aligned} \tag{2.5}$$

For the sum to make sense in (2.5), we are assuming that the dimension of $\mathscr{H}_r$ is countable. This is the case when Φ_r is constructed from the countable eigenfunction expansion of $K_r(\boldsymbol{x_r}_i, \boldsymbol{x_r}_j)$ (see [54]). □

In words, the tensor product kernel allows us to define different kernels on different attributes of our data set and to turn them into a valid kernel, that can be used in any kernel-based algorithm, like SVDD, by element-wise multiplying the Gram matrices.

Furthermore, as the constituent kernels used throughout this chapter are Gaussian RBF kernels, the tensor product kernel boils down to a multivariate Gaussian RBF kernel where each attribute vector has it own scaling factor

$$\begin{aligned} k(\boldsymbol{x}_i,\boldsymbol{x}_j) &= \prod_{r=1}^{R} \exp\Big(\frac{-\|\boldsymbol{x_r}_i - \boldsymbol{x_r}_j\|^2}{2\sigma_r^2}\Big) \\ &= \exp\Big(\sum_{r=1}^{R} \frac{-\|\boldsymbol{x_r}_i - \boldsymbol{x_r}_j\|^2}{2\sigma_r^2}\Big). \end{aligned}$$

2.3 SVDD-based Anomaly Detection for Spatio/Temporal Data

In this section we formulate our algorithm for discovering anomalies in spatio/temporal data. It extends the algorithm for discovering novelty in time series given in [46] in three ways. Firstly, we extend the data representation in [46] with spatio/temporal information – in case of time dependent data, the time stamp. Including a time stamp in the data representation is an idea that is also proposed in [82]. Secondly, we use a composite tensor product kernel to relate the time and the data part of the representation respectively. Thirdly, by giving a more general definition of the identification function that pinpoints anomalous events, we improve the ideas in [46], as will be confirmed in Section 2.5.

We formulate the algorithm in the context of monitoring variables in a production process that is time dependent. Afterwards, we point out what has to be changed in order to adapt it for more general spatial data, like the detection of anomalies in images.

Data Representation

Suppose we are given K training batches $\boldsymbol{X}^k$, $k = 1, \ldots, K$. Each of the batches can be represented as a matrix, that contains N_k rows and where each row contains the time stamp x_t and a vector $\boldsymbol{x}_\mathbf{d}$ containing the value of the d variables at that particular time, i.e.

$$\boldsymbol{X}^k = \left[x_t^k(i) \quad \boldsymbol{x}_\mathbf{d}^k(i) \right]_{i=1,\ldots,N_k}.$$

Using a single time stamp implies that the measurements of different variables are gathered at the same point in time during the production process. Nevertheless, note that different batches may have different time stamps and might vary in length.

As already mentioned in the introduction, it does not suffice to construct a global model on the time interval and compare the batches as a whole to this model. This could only respond to queries whether or not to consider the batch as an anomaly or not, but would not give us any clue where the batch has gone wrong. To accomplish the latter, we need to deduce for every sample point in time whether it is still on track or not. As SVDD needs a set of m unlabeled vectors as training set, a straightforward way to present our training batches is by extracting all sample points. In this case m no longer equals the number of batches K, but is now the number of measurements taken over all batches, i.e. $m = \sum_{k=1}^{K} N_k$.

This approach, however, has its limitations. By throwing the data in a single pile, we lose information. We lose specific relationships concerning the vicinity inside and across multiple batches. If we only take the current time stamp and the corresponding measurement into account, we will be constructing a point-wise model that merely models the data boundary and thus is incapable of detecting anomalies such as time shifts and phases that take too long. Moreover, if timings across training batches are not completely synchronised this results in a very sparse training set.

The easiest way to represent vicinity within a batch is by using a (discrete) sliding window technique and thus by incorporating measurements of previous time stamps into the attribute vector. This idea is already written down by many, specifically in literature about modeling time series [60]. We refer to [46] where the technique of unfolding a time series into a phase space using a time-delay embedding process, is already combined with one-class SVM for novelty detection in time series.

Definition 2 (Time-delay embedding process of a time series)**.** Given an embedding dimension $E \in \mathbb{N}$, a time series $x_d(i)$, $i = 1, \ldots, N$, can be converted to a set of vectors $\boldsymbol{x}_{d,E}(i)$, $i = E, \ldots, N$, where

$$\boldsymbol{x}_{d,E}(i) = \left[x_d(i-E+1) \quad x_d(i-E+2) \quad \ldots \quad x_d(i) \right].$$

As we are dealing with batches where not one variable x_d but several variables $\boldsymbol{x_d}$ are monitored, we need to apply the time-delay embedding on each of the measurements. Based on the above definition, the training data T_E we feed to the SVDD algorithm is given by

$$T_E = \left\{ \left\{ \left[x_t^k(i_k) \quad \boldsymbol{x}_{\mathbf{d},E}^k(i_k) \right]_{i_k=E,\ldots,N_k} \right\}_{k=1,\ldots,K} \right\} \tag{2.6}$$

where

$$\boldsymbol{x}_{\mathbf{d},E}^k(i_k) = \left[\boldsymbol{x}_\mathbf{d}^k(i_k-E+1) \quad \boldsymbol{x}_\mathbf{d}^k(i_k-E+2) \quad \ldots \quad \boldsymbol{x}_\mathbf{d}^k(i_k) \right].$$

Note that our representation differs from the one used in [46] because we include in the data representation the time stamp of the most recent measurement of the unfolding process. With this time stamp, samples can only be considered close to each other if they are sampled at related moments in time and if the measurement values are close to one another. Leaving out the first condition by not including the time stamp in the data representation, clearly leads to a wrong similarity measure.

For the embedding process to make sense, we assume that the overall sampling rate is more or less the same across the batches. Otherwise one must fix a set of specific capture time windows and supply data from within such a window as extra parameters, rather than just using the previous E measurements.

Time Kernel versus Data Kernel

In [47] the authors introduce the general concept of composite kernels together with two different forms, namely the direct sum kernel and the tensor product kernel. Composite kernels can be useful to combine various attributes of the data, each with different characteristics, while preserving for each attribute a distinct representation in feature space. A multivariate Gaussian RBF kernel with a different width for each attribute is a special case of a tensor product kernel but much more general tensor product kernels can be constructed by element-wise multiplying the Gram matrices of the constituent kernels for each attribute [47].

Given that the training vectors for our SVDD algorithm are of the form

$$\boldsymbol{x}_E(i) = \begin{bmatrix} x_t(i) & \boldsymbol{x}_{\mathbf{d},E}(i) \end{bmatrix}$$

the use of a tensor product kernel seems natural. In this chapter we combine two different Gaussian RBF kernels: one on the time part $x_t(i)$ and one on the measurements $\boldsymbol{x}_{\mathbf{d},E}(i)$. In the rest of the chapter we simply refer to the time kernel and the data kernel, each specified by the parameter σ_t and σ_d respectively. Note that we assume that the measurements are scaled so that we can use a single parameter σ_d for the data kernel.

The need for a different width of the kernel can be easily seen as follows. First, the time stamps might differ in scale from the other measurements, but, more importantly, vicinity relations that express closeness in time are not necessarily the same as those in value.

Furthermore, note that the batches might vary in length and do not need to be perfectly aligned. Small variations in timings will be handled by the time kernel, while small variations in the data vector will be administered by the data kernel.

In addition, if we combine both a Gaussian RBF kernel on the time stamps and one on the measured variables, we are introducing a continuous delay window that slides over the batches. It compares only those samples that are close to each other in time across the various presented batches.

Robust Anomaly Detection

Different values for the embedding dimension E lead to different representations of the time series. Methods to determine proper embedding dimensions exist in the literature on nonlinear time series analysis [34]. In this work we use another approach. As in [46], we train several SVDDs using different embedding dimensions and then apply an identification function to the output of these SVDDs.

Definition 3 (Anomalous point)**.** Given a set S of different embedding dimensions E, voting thresholds θ_S and θ_E in $[0,1]$, we say that the i^{th} sample $\boldsymbol{x}_E(i)$ is an anomalous point only when the indication function $I(i) = 1$, where

$$I(i) = \text{sgn}(\frac{1}{|S|}\sum_{E\in S} I(E,i) - \theta_S),$$
$$I(E,i) = \text{sgn}(P(E,i) - \theta_E),$$
$$P(E,i) = \frac{1}{E}\sum_{e=0}^{E-1} f_E(\boldsymbol{x}_E(i+e)),$$

and f_E is the decision function of the SVDD trained on the T_E representation. Observe that $f_E(\boldsymbol{x}_E(i)) = 0$ for $i < E$ and $i > N$.

For each measurement $\boldsymbol{x}_E(i)$, $P(E,i)$ reflects the (relative) number of windows that it is part of and that cause detection of an anomalous event. We will explain the motivation behind this choice with a simple example (see Table 2.1).

Table 2.1: Example: robust anomaly detection.

$f_3(\boldsymbol{x}_3(i))$	i	1	2	3	4	5	6	7
	1	0	–	–	–	–	–	–
	2	0	0	–	–	–	–	–
	3	1	1	1	–	–	–	–
	4	–	1	1	1	–	–	–
	5	–	–	1	1	1	–	–
	6	–	–	–	0	0	0	–
	7	–	–	–	–	0	0	0
$P(3,i)$		1/3	2/3	3/3	2/3	1/3	0/3	0/3
$I(3,i)$; $\theta_3 = 0$		1	1	1	1	1	0	0
$I(3,i)$; $\theta_3 = 0.9$		0	0	1	0	0	0	0

Suppose we are working with $E = 3$ and that $f_3(\boldsymbol{x}_3(3)) = 1$, $f_3(\boldsymbol{x}_3(4)) = 1$, $f_3(\boldsymbol{x}_3(5)) = 1$, $f_3(\boldsymbol{x}_3(6)) = 0$, $f_3(\boldsymbol{x}_3(7)) = 0$. Then $P(3,3) = 1$, $P(3,4) = \frac{2}{3}$ and $P(3,5) = \frac{1}{3}$. This is consistent with the fact that $\boldsymbol{x}_\mathbf{d}(3)$ occurs as feature attribute in three windows for all of which the decision function f_E equals 1, while $\boldsymbol{x}_\mathbf{d}(4)$ is a feature attribute in three windows, but only for two of these windows is the decision function f_E equal to 1. With $\theta_E = 0.9$, we then have that $I(3,3) = 1$, while $I(3,4) = I(3,5) = 0$, and so

Algorithm 1 Offline SVDD-based Anomaly Detection (SVND) for Temporal Data

Step 0: Parameter initialisation
Choose model parameters

1. embedding dimension range: S
2. SVDD parameter: ν
3. kernel function and parameters used by SVDD: composite Gaussian RBF kernel with (different) σ_t and σ_d, which specify the widths of respectively the time and data kernel
4. voting thresholds: θ_E and θ_S

Step 1: Model training
Given K training batches, where every batch $\boldsymbol{X}_k$, $k = 1,\ldots,K$ consists of time dependent $x_t^k(i)$ and data part $\boldsymbol{x}_{\mathbf{d}}^k(i)$, $i = 1,\ldots,N_k$

for all E in S **do**
construct set of training vectors T_E given by (2.6)
train a SVDD model on T_E using composite kernel on the time part $x_t^k(i)$ and data part of $\boldsymbol{x}_{\mathbf{d},E}^k(i)$
end for

Step 2: Anomaly detection
Given a test batch $\boldsymbol{X}$

for all E in S **do**
construct set of test vectors T_E given by (2.6)
test every ith vector for anomaly using $f_E(\boldsymbol{x}_E(i))$
check for anomaly in model using $I(E,i)$
end for
check for global anomaly using $I(i)$

only the 3rd sample is considered abnormal for embedding dimension $E = 3$. With $\theta_E = 0$, which is the approach taken in [46], the samples on the whole interval $[1,5]$ are tagged as abnormal for $E = 3$. This is less intuitive and may lead to a large number of false positives, especially when the embedding dimension E is large. When combining multiple embedding dimensions, the authors in [46] therefore rely on an all-agree voting scheme ($\theta_S = 1$) for the indication function $I(i)$ rather than on a majority voting scheme ($\theta_S = 0.5$) as in our approach (see Section 2.4).

Algorithm

We summarise the above ideas in Algorithm 1 which we use to train the model and test for anomalies. This algorithm can also be modified to be used in an online environment. In that case, we need to take into account the delay factor caused by the largest embedding dimension E in S.

There is no restriction on using this algorithm only for unidimensional temporal data. Vicinity in terms of spatial location in multidimensional spaces lies open for exploration. For example to detect anomalies inside images, one needs to replace the time kernel by a spatial kernel that takes into account the 2D pixel coordinates and generalise the indication function so that it no longer depends on a single index (i) but on the indices (i,j).

We remark that spatio/temporal data exhibit autocorrelation and heteroscedasticity. The value of each sample is therefore affected by its neighbours and the variance of the data is not uniform, but is a function of the location in time or space [75]. Therefore it is obvious that the samples are not independent and identically distributed and thus fail to meet the i.i.d. assumption. This fact implies that theoretical results obtained for SVDD or one-class SVMs, such as the PAC performance bound [66], no longer remain valid. In practice, this seems not to damage the validity of using SVDD for anomaly or novelty, as confirmed by [17, 46] and our experiments.

2.4 Experimental Setup

Despite the huge amount of data available from real production processes, there is to our knowledge no labeled, publically available benchmark data set available to (numerically) evaluate and compare the accuracy of our algorithm with existing techniques for anomaly detection. For the experiments, we will be using both (unlabeled) real and (labeled) artificial data to demonstrate the promising performance of our algorithm.

Real Data

We have applied our algorithm to real, unlabeled data coming from the production process of a chemical product – denoted as 'ProdX' for confidentiality reasons – at Dow Benelux. The data set consists of 110 batches. In each batch 6 variables are monitored as illustrated in Figure 2.1. Here we apply anomaly detection to only one of the monitored variables but the experiment can be repeated for all variables and leads to a similar performance. Out of the 110 batches we have selected 2 training sets, one consisting of 22 clean batches and one consisting of these 22 batches augmented with 3 batches which highly differ from the expected behaviour. The purpose of this augmented training set is to demonstrate the robustness of our algorithm with regard to the presence of highly erroneous batches in the training set. This will further be discussed in Section 2.5. The three sets of batches are displayed in Figure 2.2.

Artificial Data

Because the results on real, unlabeled data are difficult to quantify, we considered artificial data which is generated as follows. First we have Δ_1 zeros after which we generate a sinus function that takes Δ_2 samples to climb to 1, Δ_3 samples to reach zero, Δ_4 samples to reach -1 and Δ_5 to return to zero again, and the batch ends with one more period of Δ_6 zeros. In all batches we use discrete timestamps t ranging from 1 to $N = \sum_i \Delta_i$.

The standard distribution of the duration of each period is as follows

$$\mathbf{\Delta} = \begin{bmatrix} 100 & 75 & 75 & 75 & 100 \end{bmatrix}.$$

In order to create (small) differences in sample density and in total batch length, each entry in $\mathbf{\Delta}$ is modified by adding a uniform random noise term chosen from $[-5, 5]$.

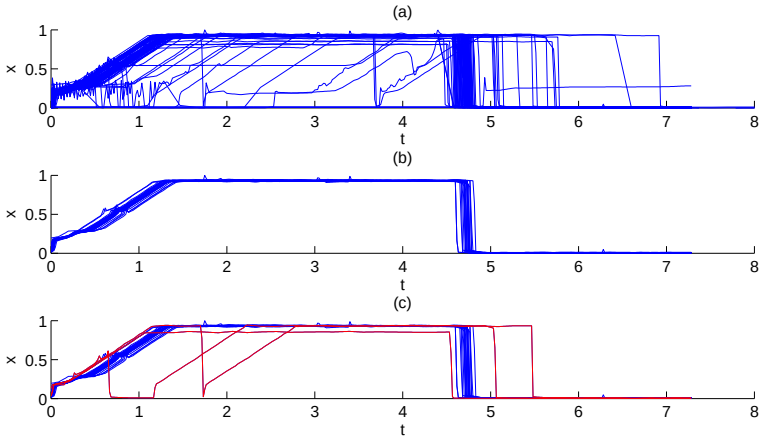


Figure 2.2: Shown are the real data monitoring a single variable x over time t: (a) all 110 batches, (b) 22 clean batches reflecting the normal scenario and (c) same 22 clean batches augmented with 3 batches that highly differ from the expected norm.

All batches are subject to additive Gaussian white noise with zero mean and standard deviation equal to 0.025. In 10 of the 20 training batches and 115 of 232 the test batches, we add bursts of extra noise (5 times the standard deviation of the white noise) to simulate anomalies. The duration d and place of this error term is chosen uniformly from respectively $[1,25]$ and $[1,N-d]$, where N denotes the length of a batch. We refer to the batches without bursts of extra noise as clean batches, and to those with extra noise as noisy batches.

The regions where some artificial errors are introduced are shaded in the subsequent figures. However, as we are dealing with one-class classification, i.e. unsupervised learning, we discard the labels during training and use them only for performance evaluation afterwards.

We repeat each experiment 10 times. Each time the artificial data set consists of 20 different training batches. The set of 232 test batches remains the same during all experiments.

Parameter Settings and Implementation Details

The parameters in Algorithm 1 are set using some heuristics. The parameters ν, σ_t and σ_d are data dependent. For the real data, we set $\sigma_t = 0.25$ and $\sigma_d = 0.05$. Because the amount of anomalies is rather small, we set $\nu = 0.05$. This allows 5% of the data to become non-bound support vectors (and thus to be considered as true anomalies). For the artificial data $\sigma_t = 100$, $\sigma_d = 0.25$ (7 time-stamps are fully compared and there is a fast decay for larger differences) while $\nu = 0.05$. For all experiments the set of embedding dimensions S consists of the odd numbers from 1 to 19, as in [46]. The voting thresholds are set to $\theta_E = 0.9$ and $\theta_S = 0.5$. On the artificial data, we will also compare this threshold combination to $\theta_E = 0$ and $\theta_S = 1$, corresponding to the settings used in [46]. We will refer to our algorithm with the

former threshold combination as SVND (majority) and to the latter as SVND (all-agree).

To construct the various Support Vector Data Descriptions, we use the fast incremental SVDD algorithm provided by DDTools [78], which we extend to use the composite tensor product kernel. As kernel we plug into this implementation a multivariate Gaussian RBF kernel which is the tensor product of a Gaussian time and a Gaussian data kernel.

All experiments are run on Sun Fire V20z nodes (dual AMD Opteron with 4 or 8 GB RAM).

2.5 Experimental Analysis

This section presents the experimental results on both real and artificial data. In order to evaluate the performance of our technique, we compare it to two baseline models used in process control [59]. The first baseline model is constructed by calculating the point-wise mean (mean(t)) and standard deviation (std(t)) over all training batches. If the new test sample with time stamp t is outside the region bounded by [mean(t) $\pm$ 3std(t)] it is considered an anomaly. The second base line model keeps tracks of the (point-wise) minimum and maximum value of the samples provided in the training batches.

Real Data

In Figure 2.3 we display the anomalies detected by our algorithm (SVND) as well as those detected by the two baseline models ([mean±3·std], [min,max]) for the test batch plotted in red. The decisions of the various algorithms are marked beneath the batch data, using the color and in the order as prescribed by the legend. In Figure 2.3a the training set consists of the 22 clean batches in Figure 2.2(b), while in Figure 2.3b the training set is the augmented training set in Figure 2.2(c). Visual inspection, due to the lack of labeled data, shows that while the performance of our algorithm is the same as that of the baseline models for the clean training set, our algorithm clearly outperforms the baseline models on the noisy training set.

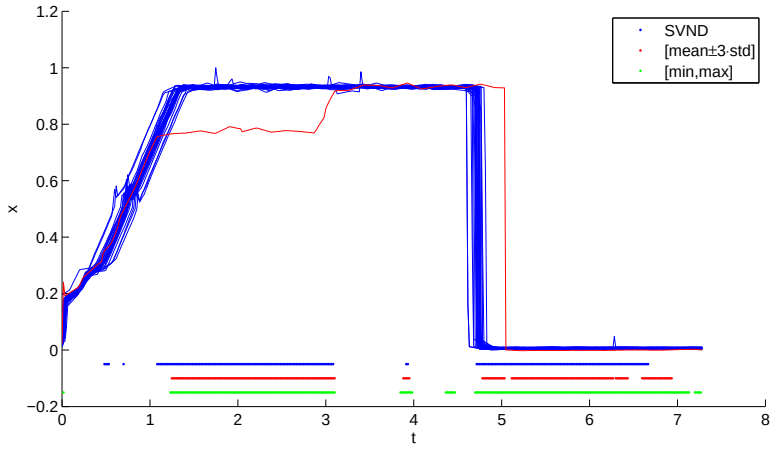
Only when zooming in, can one observe that for almost all $t \geq 5$ the test batch in Figure 3(a) falls below and outside the intervals [min,max] and [mean $\pm$ 3· std] and hence is classified as anomaly in that region of the time domain.

Artificial Data

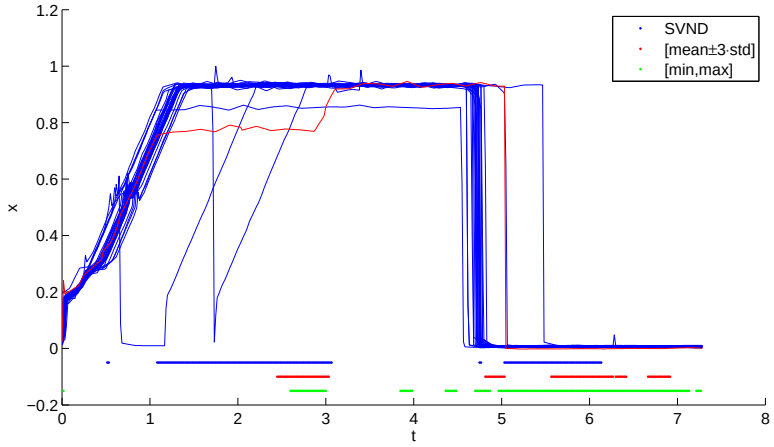
To measure the performance of our algorithm on the artificial data, we keep track of the false negative rate ($\mathscr{E}_I$), i.e. the percentage of good samples misclassified as anomalies, and the false positive rate ($\mathscr{E}_{II}$), i.e. the percentage of anomalies erroneously classified as good data. We exclude the clean batches from the calculation of the false positive rate, since there are no anomalies in the clean batches, and hence no chance of making errors of the second kind (type II errors).

Table 2.2 shows the average mean and standard deviation (over 10 runs) of the false negative and false positive rate on both training and test data.

For the baseline models we observe a negligible false negative rate on the training data and a high false positive rate on training and test data (far too many true



(a) clean training set



(b) augmented training set

Figure 2.3: Anomalies detected in real data for the test batch plotted in red by our algorithm (SVND) and two point-wise baseline models trained on 22 clean batches (a) and two additional erroneous batches (b).

anomalies are not detected). We also clearly see that SVND (majority) outperforms SVND (all-agree) with respect to the false positive rate, on both training and test batches. For the false negative rate, both perform similarly. Overall, we observe that the false negative rate of both SVND approaches is in between that of the two baseline models, but that the false positive rate of SVND (majority) is significantly better than the other three.

For our artificial problem the embedding dimension $E = 11$ yields the minimal number of false positives/negatives. From $E \geq 13$ on the error rates increase again

Table 2.2: Error rates on artificial data.

		training data		test data	
		$\mathscr{E}_I$	$\mathscr{E}_{II}$	$\mathscr{E}_I$	$\mathscr{E}_{II}$
baseline	mean std	0.00 ± 0.00	0.38 ± 0.43	0.01 ± 0.01	0.67 ± 0.24
	min max	0.00 ± 0.00	0.49 ± 0.51	0.08 ± 0.07	0.51 ± 0.26
SVND	all-agree	0.00 ± 0.00	0.27 ± 0.32	0.03 ± 0.05	0.49 ± 0.23
	majority	$\mathbf{0.00 \pm 0.01}$	$\mathbf{0.13 \pm 0.22}$	$\mathbf{0.03 \pm 0.06}$	$\mathbf{0.13 \pm 0.21}$
SVDD	$E = 1$	0.04 ± 0.05	0.25 ± 0.30	0.05 ± 0.06	0.48 ± 0.22
	3	0.01 ± 0.02	0.18 ± 0.25	0.03 ± 0.06	0.29 ± 0.24
	5	0.00 ± 0.01	0.15 ± 0.23	0.02 ± 0.05	0.21 ± 0.23
	7	0.00 ± 0.00	0.15 ± 0.24	0.02 ± 0.05	0.20 ± 0.23
	9	0.00 ± 0.01	0.16 ± 0.25	0.02 ± 0.05	0.21 ± 0.24
	11	0.01 ± 0.02	0.10 ± 0.20	0.03 ± 0.06	0.11 ± 0.21
	13	0.01 ± 0.02	0.12 ± 0.22	0.03 ± 0.06	0.13 ± 0.22
	15	0.01 ± 0.03	0.14 ± 0.26	0.03 ± 0.06	0.14 ± 0.25
	17	0.01 ± 0.02	0.15 ± 0.26	0.03 ± 0.06	0.16 ± 0.26
	19	0.01 ± 0.03	0.20 ± 0.30	0.03 ± 0.06	0.18 ± 0.28

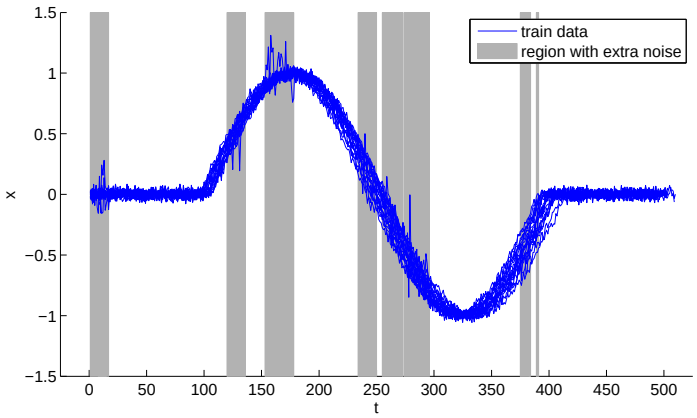
Shown are the average (mean ± std) false negative rate ($\mathscr{E}_I$) and false positive rate ($\mathscr{E}_{II}$) on training and test data for the two point-wise baseline models, SVND with two voting schemes and the SVDD models for different embedding dimensions E.

as we start to over-fit. We also remark that the all-agree results are highly correlated with the results for $E = 1$. By leaving the results for $E = 1$ out of the all-agree voting scheme, as is done in [46], its performance can be improved.

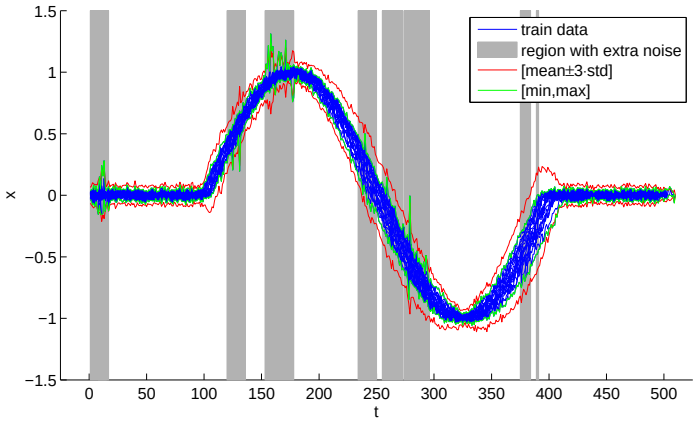
The results in Table 2.2 can be further explained by looking at Figure 2.4 and 2.5.

In Figure 2.4a, we plot a set of 20 training batches and the regions with anomalies (denoted by the shaded areas), while Figure 2.4b shows the two corresponding baseline models computed from this training set. From Figure 2.4b it is clear that the baseline models only model the boundary of the data and are sensitive to the presence of anomalies in the training set.

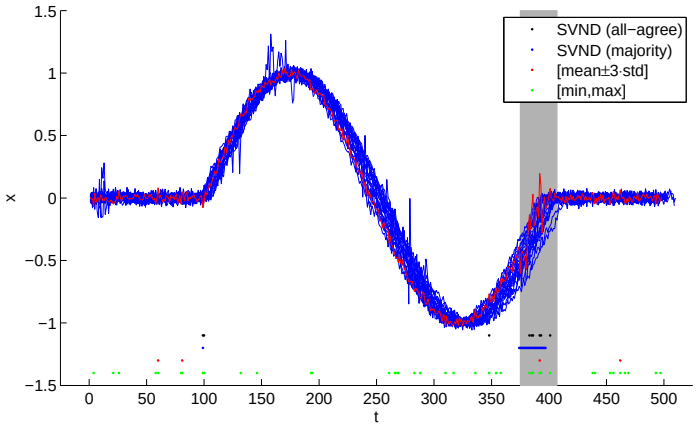
It is not possible to plot the decision boundary for SVND since it combines several SVDD models, each trained using a different embedding dimension. However, in Figure 2.5 we plot for specific embedding dimensions the bound and non-bound support vectors. This gives a rough idea of the decision boundary and provides more insight, since the bound support vectors lie on the hypersphere in feature space, while the non-bound support vectors are the samples that the SVDD algorithm considers as anomalies and fall outside the hypersphere.



(a) Training data

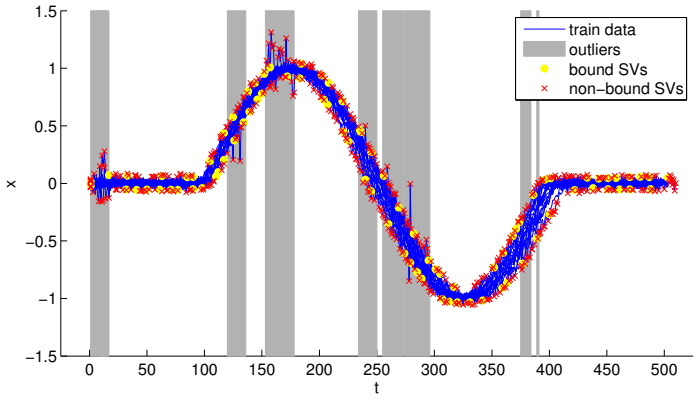


(b) Baseline models

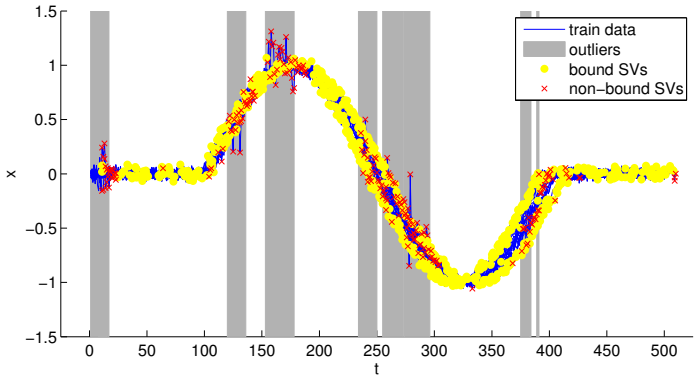


(c) Noisy test signal

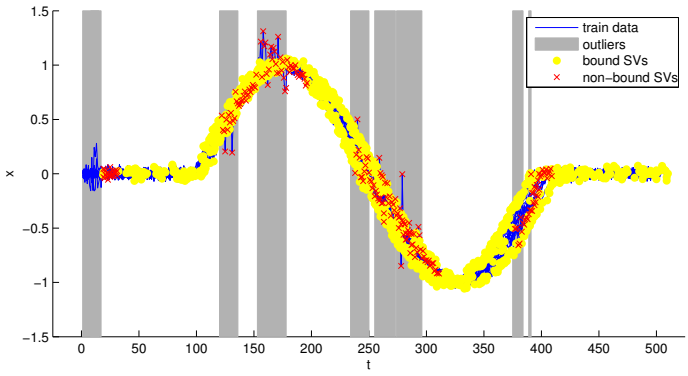
Figure 2.4: Artificial data: training data (a), baseline models (b) and the abnormalities identified on a noisy test signal (c).



(a) $E = 1$



(b) $E = 11$



(c) $E = 19$

Figure 2.5: SVDD models obtained using the training data from Figure 2.4a for different embedding dimensions. Notice that for $E = 1$ we merely model the data boundary, while for higher embedding dimensions almost all the non-bound support vectors (red crosses) lie in noisy regions.

Table 2.3: Distribution support vectors.

E	bound SVs	non-bound SVs
1	0.88 ± 0.03	4.56 ± 0.03
3	1.69 ± 0.12	4.14 ± 0.08
5	2.68 ± 0.16	3.68 ± 0.09
7	3.99 ± 0.16	3.02 ± 0.09
9	5.45 ± 0.37	2.47 ± 0.14
11	6.64 ± 0.24	2.13 ± 0.16
13	7.70 ± 0.32	2.09 ± 0.21
15	8.71 ± 0.35	2.11 ± 0.26
17	9.54 ± 0.39	2.18 ± 0.27
19	10.36 ± 0.39	2.24 ± 0.25

Shown are the percentage (mean ± std) of the training data that are bound and non-bound support vectors for different embedding dimensions E.

Figure 2.5a shows that, like the baseline models, the SVDD model constructed with $E = 1$ only strictly captures the data boundary, resulting in a high false positive rate. Also note that there is no clear distinction between the distribution of the bound and non-bound support vectors, resulting in a high number of false negatives.

In Figure 2.5b and 2.5c, we see that when the embedding dimension increases the non-bound support vectors are more capable of identifying the real anomalies present inside the training data and they no longer lie only at the boundary. If the embedding dimension gets too big ($E \geq 13$), the amount of bound support vectors is much larger than the non-bound support vectors and overfitting occurs. This results in the fact that (again) less anomalies are detected and thus a higher false positive rate on both the training and test batches.

In Table 2.3 we give the percentage of bound and non-bound support vectors for the different embedding dimensions. We see that that the amount of bound support vectors increases and reaches over 10% of the training data when we use a feature vector consisting of the current measurement and the 18 previous ones. More important, however, is the distribution of the bound versus non-bound support vectors. We see in Table 2.3 that the amount of non-bound support vectors first decreases as the embedding dimension increases, but that from $E = 15$ it increases again though in a less steep way. This tendency is also noticeable in terms of the false positive rate on both training and test data, but note that the increase starts at $E = 13$ in Table 2.2.

In Figure 2.4c we illustrate the different algorithms on a noisy signal of which several anomalies are located inside the boundaries of the training data. The figure confirms that our algorithm is the only one able to detect not only the anomalies located outside these boundaries but also the anomalies located inside of them.

To numerically illustrate the robustness of our algorithm with respect to the presence of highly erroneous batches in the training set, we now add to the training

Table 2.4: Robust test error rates.

		$\mathscr{E}_I$	$\mathscr{E}_{II}$
baseline	mean std	0.00 ± 0.01	0.79 ± 0.28
	min max	0.07 ± 0.04	0.58 ± 0.27
SVND	all-agree	0.04 ± 0.06	0.49 ± 0.23
	majority	$\mathbf{0.03 \pm 0.06}$	$\mathbf{0.11 \pm 0.21}$
SVDD	$E = 1$	0.05 ± 0.07	0.48 ± 0.22
	3	0.03 ± 0.06	0.28 ± 0.24
	5	0.03 ± 0.06	0.20 ± 0.21
	7	0.03 ± 0.06	0.19 ± 0.22
	9	0.03 ± 0.06	0.19 ± 0.24
	11	0.04 ± 0.07	0.10 ± 0.21
	13	0.04 ± 0.07	0.11 ± 0.22
	15	0.04 ± 0.07	0.12 ± 0.23
	17	0.04 ± 0.07	0.12 ± 0.24
	19	0.04 ± 0.07	0.13 ± 0.25

Shown are the average (mean ± std) false negative rate ($\mathscr{E}_I$) and false positive rate ($\mathscr{E}_{II}$) on the test data for the two point-wise baseline models, SVND with two voting schemes and the SVDD models for different embedding dimensions E, obtained with 2 highly erroneous batches, a zero and anti-phase signal, in the training data.

sets used so far for the artificial data both the zero and anti-phase signal. The results are given in Table 2.4. Comparing Table 2.2 and 2.4, we see that the performance of our algorithm remains unchanged while the two baseline models suffer, as expected, an increase of the false positive rate.

2.6 Discussion

The experiments on artificial and real data show that in the case of univariate (temporal) modeling the results are quite promising. Compared to standard techniques to monitor production processes, SVND is highly robust for noise and erroneous batches in the training data. Higher embedding dimensions allow us to step away from a strict boundary based model and to detect anomalies due to time-shifts, abnormal peaks and phases that take too long, resulting in significantly less false positives and false negatives.

It is well known that different values for the embedding dimension E lead to different representations of the batches in phase space. We opted for using an ensemble method to postpone the selection of this non-trivial parameter. The distribution of the support vectors however, seems to provide a nice heuristic to determine a good embedding dimension in the context of anomaly detection.

In the future, several tracks can be further explored. On one hand, there is a need to see how well this method scales and if it also performs well in practice on spatial data, e.g. for detecting anomalies inside images. On the other hand, the performance of our algorithm to monitor multivariate (temporal) variables needs to be further investigated.

The results of our preliminary experiments on the multivariate batches of 'ProdX' are harder to interpret. The higher dimensionality of the problem limits the use of visualisations to obtain insights in the constructed models, while the lack of labeled data leaves us without clues for starting a detailed inspection.

Moreover, we observe that scaling all the data between 0 and 1 and running SVND using a single scale parameter σ_d for the data kernel is a too simple approach. As is often necessary in the SVM approach, optimisation of the scale parameter for each input variable seems to be necessary. Also, it is clear that different time stamps may be needed to account for the fact that the various variables are measured at different time stamps. This is, however, easily accommodated in the framework we have presented.

2.7 Conclusion

In this chapter we presented a new algorithm for detecting anomalies in spatio/temporal data using one-class support vector machines. The tensor product combination of a time (or spatial) kernel and a data kernel, enables us to exploit vicinity relations in both time (or space) and data. A more general voting scheme combining an ensemble of SVDDs enables us to incorporate historical information provided by different time-delay embeddings without suffering loss of robustness or overfitting. In case of temporal data, our algorithm detects anomalies due to time-shifts, abnormal peaks and phases that take too long and has a significantly better false positive and false negative rate than other standard techniques and than the approach presented in [46].

CHAPTER 3

Automated Vandalism Detection in Wikipedia

Since the end of 2006 several autonomous bots are, or have been, running on Wikipedia to keep the encyclopedia free from vandalism and other damaging edits. These hand-crafted rule-based systems, however, are far from optimal and need to be improved to relieve the human editors from the burden of manually reverting such edits.

We investigate the possibility of using machine learning techniques to build an autonomous system capable to distinguish vandalism from legitimate edits. We highlight the results of a step in this direction by applying two off-the-shelf classifiers using a straightforward feature representation.

Despite the promising results, this study also reveals that elementary features, which are also used by the current approaches to fight vandalism, will not suffice to build such a system. They will need to be accompanied by additional information which, among other things, incorporates the semantics of a revision.

This chapter is based on work published as [71]:
K. Smets, B. Goethals, and B. Verdonk. Automatic vandalism detection in Wikipedia: Towards a machine learning approach. In *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy (WikiAI), Chicago, IL*, pages 43–48, 2008.

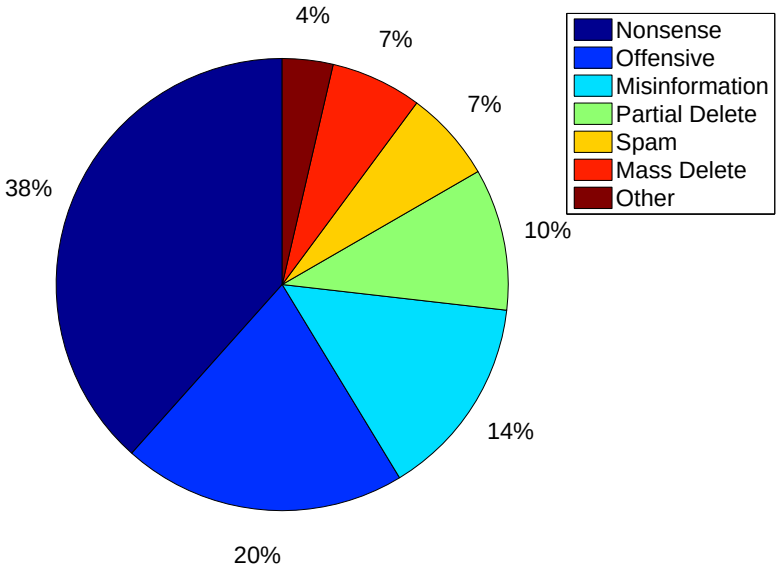


Figure 3.1: Categories of typical acts of vandalism together with the percentage of occurrence as empirically determined by Priedhorsky et al. [64].

3.1 Introduction

Since the inception of Wikipedia in 2001, the free encyclopedia, which is editable by anyone, has grown rapidly to become what it is today: one of the largest sources of adequate information on the Internet. This popularity translates itself to an ever growing large amount of articles, readers consulting them, editors improving and extending them ... and unfortunately also in the number of acts of vandalism committed a day. By vandalism we understand every edit that damages the reputation of articles and/or users of Wikipedia. Priedhorksy et al. [64] provide a survey of the typical categories of damages together with an empirically determined likeliness of occurrence. Key examples, in decreasing order of appearance, include among others introducing nonsense, offenses or misinformation, the partial deletion of content, adding spam (links), and mass deletion of an article. In Figure 3.1 we show the percentage of occurrence of these typical acts of vandalism.

To fight vandalism, Wikipedia relies on the good faith of its users that accidentally discover damaged articles and, as in practice turns out, on the time-consuming efforts of its administrators and power users. To ease their job, they use special tools like Vandal Fighter to monitor the recent changes and which allow quick reverts of modifications matching regular expressions that define bad content or are performed by users on a blacklist. Since the end of 2006 some vandal bots, computer programs designed to detect and revert vandalism have seen the light on Wikipedia. Nowadays the most prominent of them are ClueBot and VoABot II. These tools are built around the same primitives that are included in Vandal Fighter. They use lists of regular expressions, and consult databases with blocked users or IP addresses to keep legitimate edits apart from vandalism. The major drawback

of these approaches is the fact that these bots utilise static lists of obscenities and 'grammar' rules which are hard to maintain and easy to deceive. As we will show, they only detect 30% of the committed vandalism. So there is certainly need for improvement.

We believe this improvement can be achieved by applying machine learning and natural language processing (NLP) techniques. Not in the least because machine learning algorithms have already proven their usefulness for related tasks, such as intrusion detection, spam filtering for email, as well as spam filters for weblogs.

The remainder of this chapter is organised as follows. First, we motivate in Section 3.2 the use of machine learning to detect vandalism automatically, followed by a brief overview of related work in Section 3.3. Next, we complement in Section 3.4 the most recent vandalism studies by discussing the performance results of the vandal fighting bots currently active on Wikipedia. Thereafter, we describe in Section 3.5 the setup of the preliminary machine learning experiments using a Naive Bayes classifier and a compression based classifier on the same features that serve as raw input for those bots and we analyse the results in Section 3.6. Finally, we outline approaches to investigate next in Section 3.7 and we formulate conclusions in Section 3.8.

3.2 Vandalism Detection and Machine Learning

The particular task to detect vandalism is closely related to other anomaly detection problems in computer security: intrusion detection or filtering out spam from mailboxes and weblogs. It is a specific kind of web-defacement, but as the accessibility allows anyone to contribute, there is no need for crackers breaking into systems. We can see it as a form of content-based access control, where the integrity constraint on Wikipedia enforces that "All article modifications must be factual and relevant" [26]. Moreover, solutions to automatically detect vandalism need to cope with intrinsic characteristics shared among these anomaly detection problems. We need to deal with an imbalanced and ever changing class distribution as the normal edits outnumber vandalism and both vandalism and legitimate edits are likely to change, due to respectively the adversarial environment and the rise of new articles or formatting languages.

Machine learning provides state of the art solutions to closely related problems. We put two techniques from the world of spam detection to the test. On one hand we use a well-known Naive Bayes classifier and on the other hand we employ, as results from Naive Bayes models are significantly improved by state-of-the-art statistical compression models [8], a classifier based on probabilistic sequence modeling.

Although we are aware that we will not be capable of identifying all types of vandalism (e.g. detecting misinformation in the pure sense is regarded as impossible without consulting external sources of information), we believe that machine learning might cope with this interesting, challenging and far from trivial problem.

3.3 Related Work

Wikipedia has been subject to a statistical analysis in several research studies. Viégas et al. [84] make use of a visualisation tool to analyse the history of Wikipedia articles. With respect to vandalism in particular, the authors are able to (manually) identify

mass addition and mass deletion as jumps in the history flow of a page. Buriol et al. [11] describe the results of a temporal analysis of the Wikigraph and state that 6 percent of all edits are reverts and likely vandalism. This number is confirmed by Kittur et al. [35] in a study investigating the use of reverting as the key mechanism to fight vandalism. The authors also point out that only looking for reverts explicitly signaling vandalism is not strict enough to find evidence for most of the vandalism in the history of articles. In a recent study, Priedhorsky et al. [64] categorise the different types of vandalism and their occurrence rate in a subset of 676 revision chains that were reverted. They confirm that reverts explicitly commented form a good approximation to spot damages, with a precision and recall of respectively 77% and 62%. Our work complements this last one, as we investigate a yet more recent version of the English Wikipedia history, and also analyse the decisions made by two bots. We also try to respond to the authors' request to investigate the automatic detection of damage. The authors believe in intelligent routing tasks, where automation directs humans to potential damage incidents but where humans still make the final decision.

There is strong cross-pollination possible between Wikipedia and several research areas. Wikipedia can benefit from techniques from the machine learning, information retrieval and NLP domains in order to improve the quality of the articles. Adler and de Alfaro [1] build a content-driven system to compute and reflect the reputation of authors and their edits based on the time span modifications remain inside an article. Priedhorky et al. [64] use a closely related measure but they do not take into account the lifetime but the expected viewing time to rate the value of words. Rassback et al. [65] explore the feasibility of automatically rating the quality of articles. They use a maximum entropy classifier to distinguish six quality classes combining length measures, wiki specific measures (number of images, in/out links ...) and commonly used features to solve NLP problems (part-of-speech usage and readability metrics). Our problem to detect damages is related to their work in the sense that we need to rate the quality of a single revision instead of the whole article. The cross-pollination also holds for the other way around as machine learning, information retrieval and NLP can benefit from the use of Wikipedia. Gabrilovich and Markovitch [19] use a semantic interpreter built using articles from Wikipedia which is capable of measuring the semantic relatedness between text documents.

Recently, Potthast et al. [63] also use machine learning to detect vandalism in Wikipedia. Compared to their work, we have a larger, auto-labeled data set, use different classifiers, and most importantly, use different features. We aim to summarise an edit by focusing on the difference between the new and old version of an article, while Potthast et al. developed a feature set consisting of 16 numerical features that characterise their insights after manually analysing 301 cases of vandalism. Note, that the first curated vandalism corpus *Webis-WVC-07* [62] is smaller than the one we auto-labeled and its class distribution is not representative. We will however use this corpus in an additional experiment to further evaluate our trained classifiers.

3.4 Performance Analysis of Vandal Fighting Bots on Wikipedia

In this section we complement the work in [64] by analysing the results of the vandal fighting bots on one hour of data from the English version of Wikipedia. We show that there is still significant room for improvement in the automatic detection of vandalism. Furthermore, we provide additional evidence that the labeling procedure based on edit reverts, is quite sound. Next, we introduce the Simple English Wikipedia and present the results of a modified version of ClueBot on this data set, which we also use in our machine learning experiments later on. We start however with a short introduction to ClueBot's inner working.

ClueBot

ClueBot [13] uses a number of simple heuristics to detect a subset of the types of vandalism shown in Figure 3.1. First, it detects page replaces and page blanks relying on an auto-summary feature of MedaWiki software. Next, it categorises mass delete, mass addition and small changes based on absolute difference in length. For the last three types, vandalism is determined by using a manually crafted static score list with regular expressions specifying the obscenities and defining some grammar rules which are hard to maintain and easy to by-pass. Negative scores are given to words or syntactical constructions that seem impossible in good articles, while wiki links and wiki transcludes are considered as positive. The difference between the current and the last revision is calculated using a standard *diff* algorithm. Thereafter, the inserted and deleted sentences are analysed using the score list and if this value exceeds a certain threshold vandalism is signaled. ClueBot further relies on the user whitelist for trusted users and increases its precision by only reverting edits done by anonymous or new users.

Obtaining labeled data

We restrict ourselves to the recent changes of pages from the main namespace (0), the true encyclopedic articles, and ignore revisions from user or talk and discussion pages of the (Simple) English Wikipedia.

All revision data is automatically labeled by matching revision comments to regular expressions that signal a revert action, i.e. an action which restores a page to a previous version. This approach closely resembles the identification of the set of revisions denoted in [64] as Damaged-Loose, a superset of the revisions explicitly marked as vandalism (Damaged-Strict). The different labeling functions are listed in Table 3.1.

While labeling based on commented revert actions is a good first order approximation, mislabeling cannot be excluded. If we regard vandalism as the positive class throughout this chapter, then there will be both false positives and false negatives. The former arises when reverts are misused for other purposes than fighting vandalism like undoing changes without proper references or prior discussion. The latter occurs when vandalism is corrected but not marked as reverted in the comment, or when vandalism remains undetected for a long time. Estimating the number of mislabelings is very hard and manual labeling is out of question, considering the vast amount of data.

Table 3.1: Labeling functions

revert	revisions in between pairs of identical versions of an article

vloose	comments probably indicating vandalism reversion
	`(^revert\ to.+using)`
	`(^reverted\ edits\ by.+using)`
	`(^reverted\ edits\ by.+to\ last\ version\ by)`
	`(^bot\ -\ rv.+to\ last\ version\ by)`
	`(-assisted\ reversion)`
	`(^(revert(ed)?|\verb|rv).+to\ last)`
	`(^undo\ revision.+by)`

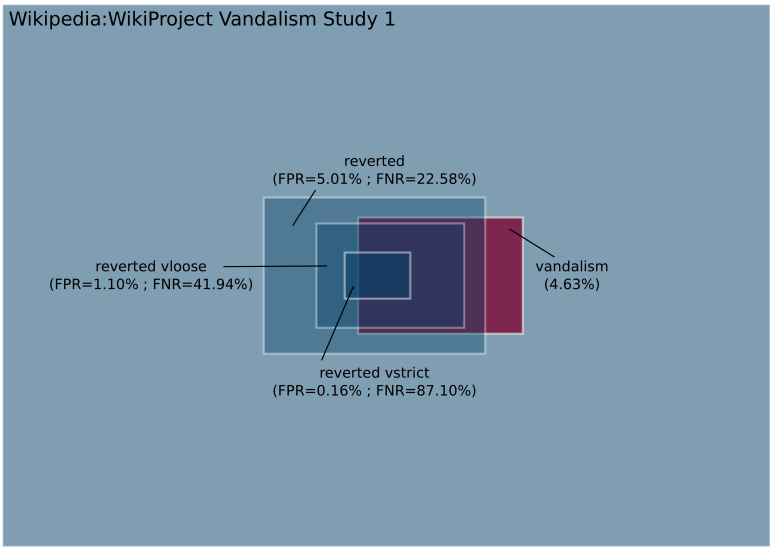
vstrict	comments almost definitely indicating vandalism reversion
	`(\brvv)`
	`(\brv[/ ]v)`
	`(vandal(?!proof|bot))`
	`(\b(rv|rev(ert)?|rm)`
	`\b.*(blank|spam|nonsense|porn|mass\sdelet|vand))`

All revision data is automatically labeled by matching revision comments to regular expressions that signal a revert action, i.e. an action which restores a page to a previous version.

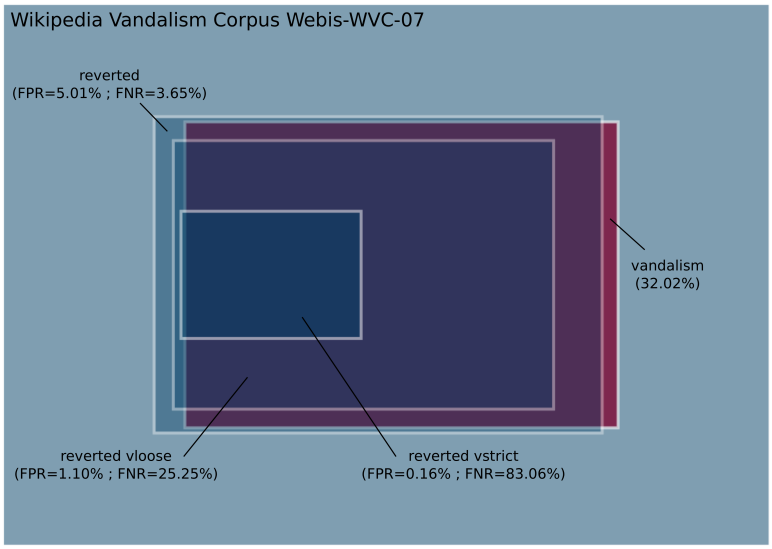
Table 3.2: Webis Wikipedia Vandalism Corpus

	legitimate	vandalism	
WP:WPVS1	639	31	(4.63%)
Webis-WVC-07	940	301	(32.0%)

Note, that only the Wikipedia:WikiProject Vandalism Study (*WP:WPVS1*) subset of the Webis Wikipedia Vandalism Corpus (*Webis-WVC-07*) contains a proper distribution of legitimate versus vandalised edits.



(a) *WP:WPVS1*



(b) *Webis-WVC-07*

Figure 3.2: Graphical representation of the error rates of the labeling functions (reverted, vloose, vstrict) listed in Table 3.1 on the Webis Wikipedia vandalism dataset (*Webis-WVC-07*) [62], which is a superset of the first Wikipedia:WikiProject Vandalism Study (*WP:WPVS1*).

Table 3.3: Edit statistics on English Wikipedia (2008.03.01).

	1 hour		5 hours	
legitimate	6944		28 312	
reverted	323		1 926	
mislabeled	26	(8.00%)	n.a.	
ClueBot	68	(22.89%)	349	(18.12%)
VoABot II	33	(11.11%)	154	(8.00%)

Shown are the number of legitimate and reverted (vloose) edits collected during 1 and 5 hours on the first of March 2008, and the number of reverts are made by ClueBot or VoABot II (the percentages reflect the recall). For the first hour we manually tracked the mistakes the auto-labeling of the reverted edits made and the false positive rate is listed between parentheses.

Figure 3.2 visualises the error rates of the labeling functions listed is Table 3.1 on the Webis Wikipedia vandalism dataset (*Webis-WVC-07*) [62], which is a superset of the first Wikipedia:WikiProject Vandalism Study (*WP:WPVS1*), listed in Table 3.2. Note that class distribution in Figure 3.2b is not representative and as a consequence the error rates, and the overlapping areas, are less meaningful. On the other hand, looking at the area of both false positives and false negatives in Figure 3.2a reveals that the labeling procedure based on signaled reverts (reverted vloose) is quite sound as the area is almost the same.

English Wikipedia (enwiki)

Table 3.3 summarises our analysis of the first hour of data from the first of March 2008 (00:00:00 - 00:59:59). From the total of 6944 revisions during the first hour, 4.65% are considered vandalism. Manual inspection demonstrates that of these 323, 11 are mislabeled as vandalism and for 15 others we are in doubt. So in the worst case we have to cope with a false positive rate of 8%.

Of the correctly labeled acts of vandalism 68 are identified by ClueBot and 33 by VoABot II, the two active vandal fighting bots on Wikipedia nowadays. Together this corresponds to a recall of 33%. Hence the bulk of the work is still done by power users and administrators. All vandalism identified by the two bots is true vandalism, and as such the precision during this one hour is 100%.

Priedhorsky et al. identify in [64] that around 20% of their labeled data is misinformation, a number confirmed by our manual inspection. Even disregarding those, the above analysis reveals there is much room for improvement w.r.t. the recall.

Empirical analysis on a data set including the next four hours, see Table 3.3, shows that these numbers are multiplied by a factor 5 and thus remain invariant.

Table 3.4: Size (Simple) English Wikipedia

	pages	revs	xml.bz2
enwiki	11 405 052	167 464 014	133.0 GB
simplewiki	53 449	499 395	88.7 MB

Size (Simple) English Wikipedia expressed in terms of the total number of pages, revisions and the compressed file size of the pages-meta-history files available at `http://download.wikimedia.org`.

Table 3.5: Edit statistics on Simple Wikipedia

period	nr (%) of vandalism	revs	pages
2003 - 2004	21 (1.12)	1 870	784
2004 - 2005	276 (2.03)	13 624	2541
2005 - 2006	2 194 (5.60)	39 170	6626
2006 - 2007	12 061 (8.33)	144 865	17 157
2007 - ...	12 322 (6.96)	177 165	22 488
2003 - ...	26 874 (7.13)	376 694	28 272

Shown are the estimated number (and percentage) of vandalised revisions together with the number of revisions and pages from the main namespace in Simple English Wikipedia using the dump made available on 2007.09.27.

Simple English Wikipedia (simplewiki)

As a proof of concept and because of storage and time constraints, we run the preliminary machine learning experiments on Simple English Wikipedia, a user-contributed online encyclopedia intended for people whose first language is not English. This encyclopedia is much smaller in size compared to the standard English Wikipedia as shown in Table 3.4. There are no bots in operation that try to remove spam or vandalism. Nevertheless the articles are also subject to vandalism, which often last longer as fewer readers and users are watching the pages.

We work with the dump from 2007.09.27 and again we only consider the main articles disregarding pages from other namespaces. Labeling using the same procedure, as outlined in Section 3.4, shows that the amount of vandalism, as we see in Table 3.5, is fairly stable and comparable with the percentages on English Wikipedia shown in Table 3.3.

As a reference, we provide the performance of a modified version of ClueBot on the simplewiki data set in Table 3.6. We use our own implementation based on the source code of the one running at enwiki, with that difference that we only consider the heuristics to detect vandalism and do not take into account the dynamic user whitelist.

Table 3.6: Emperical performance of ClueBot

	acc	pre	rec	F_1
2003 - 2004	97.52	12.50	4.76	6.89
2004 - 2005	97.22	25.77	9.05	13.40
2005 - 2006	93.46	41.85	7.61	12.88
2006 - 2007	91.44	62.07	13.06	21.58
2007 - ...	93.20	63.81	17.74	27.77
2003 - ...	92.70	61.14	14.72	23.72

Shown are the accuracy, precision, recall and F_1 of ClueBot (without user whitelist) on Simple English Wikipedia.

Compared to the edit statistics shown in Table 3.3, we notice in Table 3.6 a drop in both precision and recall. The former can possibly be explained by not using the dynamic user white list, while the fact that the static score list of the ClueBot is manually tailored towards the English Wikipedia could explain the drop in recall. A more thorough study, including manually analysing the decisions of the ClueBot, is required before we can further explain the decreased performance.

3.5 Experimental Setup

In this section, we will discuss the setting for our machine learning experiment conducted on the *simplewiki* data set, the Simple English version of Wikipedia and on the *Webis-WVC-07* corpus. We first consider the data representation. Thereafter we give a brief description of two learning algorithms put to test: a Naive Bayes classifier on bags of words (BOW) and a combined classifier built using probabilistic sequence modeling [8], also referred to in the literature as statistical compression.

Revision Representation

In this case study we use the simplest possible data representation. As for ClueBot and VoABot II, we extract raw data from the current revision and from the history of previous edits. This first step could be seen as making the static scoring list of ClueBot dynamic. This should provide a baseline for future work. In particular, for each revision we use its text, the text of the previous revision, the user groups (anonymous, bureaucrat, administrator ...) and the revision comment. We also experimented with including the lengths of the revisions an extra feature. The effect on overall performance is however minimal and thus we discarded them in this analysis. Hence the focus lies here more on the content of an edit.

As the modified revision and the one preceding it differ slightly, it makes sense to summarise an edit. Like ClueBot, we calculate the difference using the standard *diff* tool. Processing the output gives us three types of text: lines that were inserted, deleted or changed. As the changed lines only differ in some words or characters

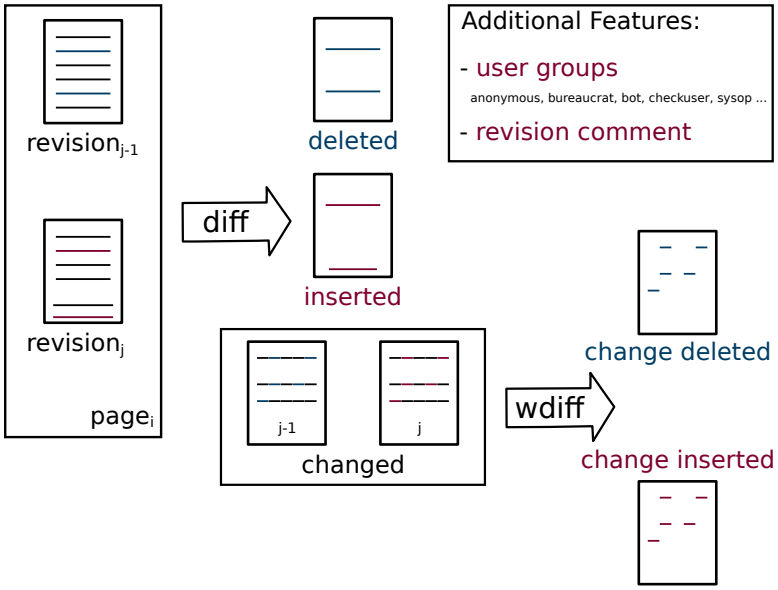


Figure 3.3: Extraction process of the straightforward revision representation.

from each other, we again compare these using *wdiff*. Basically, this is the same as what users see when they compare revisions visually using the MediaWiki software. Figure 3.3 summarises this straightforward revision representation, while Table 3.7 gives a censored example of the feature representation used throughout this chapter, applied to a vandalised revision.

Naive Bayes

As a first attempt we use the Naive Bayes implementation from the 'Bow' toolkit [51] as learning mechanism. This tool treats each feature as a bag of words and uses Porter's stemming algorithm and stop word removal to decrease the size of the feature space. Next, we train a Naive Bayes classifier on each of the features separately. Our final classifier combines the results of the individual classifiers by multiplying the obtained probability scores.

Probabilistic Sequence Modeling

Probabilisitic sequence modeling (PSM) forms the foundation of statistical compression algorithms. The key strength of compression-based methods is that they allow constructing robust probabilistic text classifiers based on character-level or binary sequences, and thus omit tokenisation and other error-prone pre-processing steps. Nevertheless, as clearly stated by [68], they are not a "parameter free" silver bullet for feature selection and data representation. In fact they are concrete similarity measures within defined feature spaces. Commonly used statistical compression algorithms are dynamic Markov compression (DMC) and prediction by partial matching (PPM), both described in detail by [8]. Basically these are n-gram

Table 3.7: Example Revision Representation

delete	Vandalism is almost always a crime; different types of vandalism include: graffiti, smashing the windows of cars and houses, and rioting. {{stub}}
insert	Being *** is almost always a crime; different types of **** *** include: ***** style.
change delete	Vandalism property vandal graffiti website vandals funny attention vandal Vandals
change insert	******* of as *** **** *** **** site **** *** *** *** ****** ***_*******
comment	
user group	anonymous

Shown is the censored feature list of revision 29853 from the Vandalism page in Simple English Wikipedia.

models where weights are implicitly assigned to the coordinates during compression. Empirical tests, in above references, show that compression by DMC and PPM outperforms the explicit n-gram vector space model due to this inherent feature weighting procedure. For the implementation we use PSMSlib [7], which uses the PPM algorithm.

During the training phase a compression model M_c^f is built [8] for each feature f in Table 3.7 and for each class c (vandalism or legitimate). The main idea is that sequences of characters generated by a particular class will be compressed better using the corresponding model. In theory, an optimal compression can be achieved if one knows the entropy given that model. In order to classify a revision r, we estimate for each of its feature values x the entropy H by calculating,

$$H_c^f(r) = \frac{1}{|x|} \log \prod_{i=1}^{|x|} p(x_i | x_{i-k}^{i-1}, M_c^f),$$

where $p(x_i | x_{i-k}^{i-1}, M_c^f)$ is the probability assigned by model M_c^f to symbol x_i given its k predecessors. In order to score the revision, we combine all features by summing over the entropies,

$$S_c(r) = \sum_f H_c^f(r)$$

and then calculating the log ratio

$$S(r) = \log \frac{S_{van}(r)}{S_{leg}(r)}.$$

If the value S exceeds a prespecified threshold, default 0, we assign the revision to the vandalism class otherwise we consider it as legitimate. The threshold parameter trades off the precision and the recall.

Evaluation Setup

To evaluate our machine learning experiments we use 60% of the labeled *simplewiki* data for training and the remaining 40% for evaluation purposes. We do not aim to statistically analyse the different approaches but use it more as a guide to conduct our search towards a machine learning based vandalism detection tool.

We will compare all classifiers using accuracy ($\frac{tp+tn}{tp+tn+fp+fn}$), precision ($\frac{tp}{tp+fn}$), recall ($\frac{tp}{tp+fn}$) and F_1 score ($2\frac{precision \cdot recall}{precision+recall}$). As these measures are highly dependent on the choice of the threshold parameter, we also include ROC and precision/recall curves to compare the overall classification performance.

For reference purposes we also include the results of the classifiers, built using all auto-labeled *simplewiki* data and evaluate these on the Webis-WVC-07 corpus, the first publically available curated data set for detecting vandalism.

3.6 Experimental Analysis

In this section we analyse the results of the two attempts to put machine learning to work on the Simple English data set and Webis Wikipedia Vandalism corpus.

Table 3.8 shows the results on the *simplewiki* test set of the final Naive Bayes classifier taking into account either the revision diff features as bags of words only or including the user group information together with revision comments. While the precision in these tables is almost the same as in Table 3.6, a significant increase can be noticed in terms of recall and F_1, especially when including user group information and comment.

Table 3.9 shows the results on the whole data set of the classifiers trained using different sets of features and provides insight in the influence of the features.

As expected, we see that the '(change) delete'-feature contributes little more than noise, while the 'change insert' is the most decisive factor. Next, we observe a seemingly important contribution of the 'change delete'-feature with respect to the recall. This may be due to the fact that some pages are vandalised more than others. It is, however, not a decisive feature as it contributes little to the overall result in terms of precision.

The domination of the 'user group'-feature on the recall can be easily explained by combining the facts that anonymous users commit most of the vandalism, but that their overall legitimate contribution to Wikipedia is rather small.

Note that the Naive Bayes classifier on all features, as shown by the last line in Table 3.9, the recall is higher but at the same time there is a drop in the precision.

Table 3.9 shows the overall performance of the classifier build using probabilistic sequence modeling together with the results of the individual models on the same

Table 3.8: Performance of Naive Bayes

	diff only				all features			
	acc	pre	rec	F_1	acc	pre	rec	F_1
2003 - 2004	97.48	40.00	44.44	42.10	97.94	50.00	44.44	47.05
2004 - 2005	96.48	30.07	36.03	32.78	96.35	29.37	37.83	33.07
2005 - 2006	92.35	37.01	29.41	32.78	91.65	34.27	34.39	34.33
2006 - 2007	92.66	69.75	32.66	44.49	92.61	61.61	48.00	53.96
2007 - ...	93.10	59.49	19.60	29.48	93.42	59.11	34.53	43.59
2003 - ...	93.03	61.66	25.03	35.61	93.14	58.82	36.94	45.38

Shown is, for every year, the accuracy, precision, recall and F_1 scores of Naive Bayes on Simple English Wikipedia using only the revision diff features in a bag of words, and including user group information and revision comments.

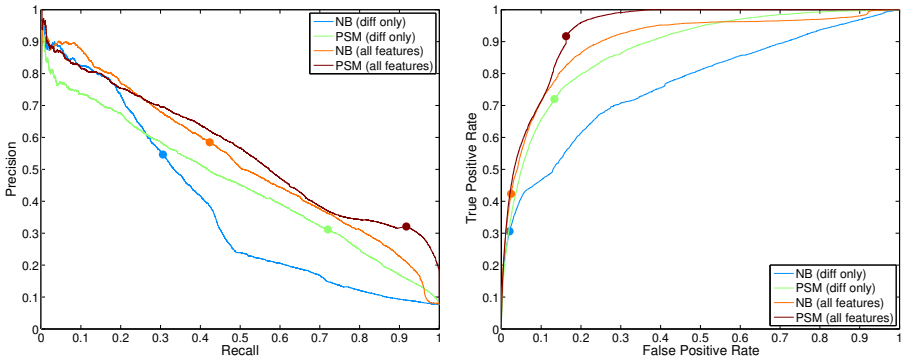
Table 3.9: Results individual classifiers on the *simplewiki* dataset.

	NB				PSM			
	acc	pre	rec	F_1	acc	pre	rec	F_1
delete	86.18	14.76	28.13	19.36	15.68	08.09	95.67	14.93
insert	95.85	26.36	26.70	26.53	50.31	12.74	92.81	22.41
change del	50.02	10.79	53.07	17.94	28.91	08.05	78.67	14.61
change ins	90.68	64.86	20.68	31.36	50.28	11.77	83.62	20.64
	93.03	61.66	25.04	35.61	85.54	31.17	72.01	43.51
comment	87.29	23.60	28.94	26.00	79.78	26.67	92.33	41.38
user group	84.44	31.02	83.19	45.20	84.60	31.71	85.98	46.33
	93.14	58.82	36.94	45.38	84.36	32.09	91.71	47.55

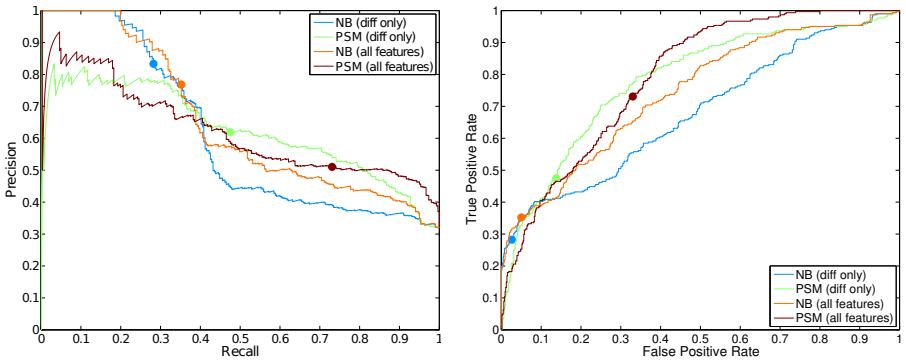
Shown are the accuracy, precision, recall and F_1 scores for the overall classifiers built using Naive Bayes and Probabilistc Sequence Modeling, together with the results of the models built for the individual models on the *simple wiki* test set.

simplewiki test set. Interesting to note is that the recall is much higher, but that the precision drops unexpectedly. We lack a plausible explanation for this strange behaviour, but the effect can be diminished by setting the threshold parameter to a score higher than zero.

This is shown in Figure 3.4, where we plot the precision/recall and receiver operating characteristic curves for varying thresholds for the probabilistic sequence models and for the Naive Bayes models, both with and without user groups and



(a) Precision/Recall curve – *simplewiki* (b) ROC curve – *simplewiki*



(c) Precision/Recall curve – *Webis-WVC-07* (d) ROC curve – *Webis-WVC-07*

Figure 3.4: Precision/Recall and ROC curves: Naive Bayes versus Probabilistic Sequence Modeling for revision diff features with(out) user group and comment on both the *simplewiki* dataset and the *Webis-WVC-07* corpus. The dots corresponds to the default classifiers as used in Table 3.9.

comments and both *simplewiki* and *Webis-WVC-07* datasets. The marks show the results when the log ratio threshold is equal to 0. The tendency is that, despite the worse behavior shown in Table 3.9, the overall accuracy measured in term of precision and recall is better for the compression based models than for the bag of words model using Naive Bayes.

Figure 3.5 summarises and compares the overall performance of ClueBot with the Naive Bayes and Probabilistic Sequence Modeling classifiers, including all features or only using the *diff* features.

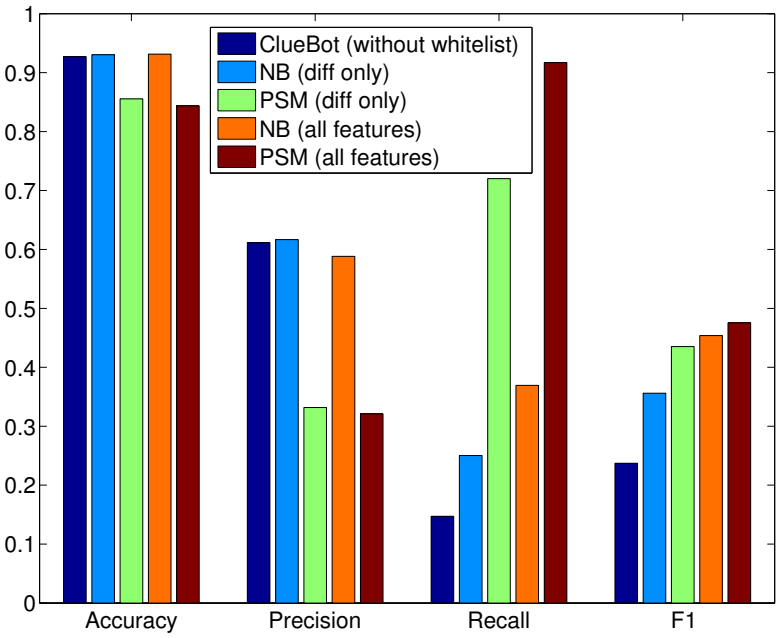


Figure 3.5: ClueBot compared to Naive Bayes and Probabilistic Sequence Modeling for revision diff features with(out) user groups and comment on *simplewiki*.

3.7 Discussion

Experiments demonstrate that, by applying two machine learning algorithms, a straight forward feature representation and using a set of noisy labeled examples, the rate to detect vandalism of the actual running bots can be improved. We feel confident that this study is merely a starting point and that there is much room for improvement.

To boost the overall performance we will need additional information. A next step might be to combine the combine the ideas from [1] and [19] to enhance the feature representation. We might rebuild their explicit semantic interpreter and use it for semantic comparison between the current modified revision and the previous versions of an article. We could compare the concepts related to text inserted and deleted, and weight these features using respectively the authority of authors and the value of words expressed in text life or expected viewing rate.

We believe that incorporating weighted semantics derived from explicit semantic analysis, as described by [19], is necessary to improve overall performance. The intuition is that the semantics of offenses, nonsense and spam are likely to differ from the semantics of the revised article and hence are an important feature for classification. Moreover, we believe that the 'text deleted'-feature contains more information than is apparent from the current results, where it appears to be merely a noise factor. To exploit the usefulness of this feature, we will take into account its effect on the semantic level by measuring the text life, i.e. the value of the deleted words, as suggested by [1].

Moreover, since the user groups play a dominant role in reducing the false positive rate, a more fine-grained feature reflecting the authority of authors [1] could be included to improve the precision of the classifiers and might be an alternative to the black-list currently used in the vandal fighting bots.

Although nowadays large manually labeled multi-lingual vandalism corpora are available for research purposes [61], using a set of noisy labeled provides still a valuable alternative to bootstrap classifiers using co-training, trained on different and more diverse sets of features, e.g. the ones used by Potthast et al. [63]. The labeling technique based on the reverted-vloose revisions used in this chapter however could be improved in several ways. First, it might be better, to start from the reverted-vstrict revisions, since these comments almost definitely indicate reversion of vandalism. Moreover, it should be possible to extract other templates, and improve the current ones, indicating reverts that cleanup vandalism, using pattern extraction techniques [9].

3.8 Conclusion

To the best of our knowledge, we were among the first to try machine learning techniques to answer the need of improving the recall of current hand-crafted rule-based systems, which are only capable of identifying 30% of all vandalism. We demonstrated that the detection rate of the current autonomous bots to fight vandalism on Wikipedia can be improved employing a machine learning approach relying on a straightforward feature representation and a set of noisy labeled training examples.

CHAPTER 4

Identifying and Characterising Anomalies in Transaction Data

In many situations there exists an abundance of normal examples, but only a handful of anomalies. In this chapter we show how in binary or transaction data such rare cases can be identified and characterised.

Our approach uses the Minimum Description Length principle to decide whether an instance is drawn from the training distribution or not. By using frequent itemsets to construct this compressor, we can easily and thoroughly characterise the decisions, and explain what changes in an example would lead to a different verdict. Furthermore, we give a technique through which, given only a few anomalous examples, the decision landscape and optimal boundary can be predicted—making the approach parameter-free.

Experimentation on benchmark and real data shows our method provides very high classification accuracy, thorough and insightful characterisation of decisions, predicts the decision landscape reliably, and can pinpoint observation errors. Moreover, a case study on real MCADD data shows we provide an interpretable approach with state-of-the-art performance for screening newborn babies for rare diseases.

This chapter is based on work published as [73]:
K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 804–815, 2011.

4.1 Introduction

In many situations there is an abundance of samples for the normal case, but no, or only a handful, anomalies. Examples of such situations include, intrusion detection [30, 38], screening for rare diseases [5, 31], monitoring in health care [21] and industry [72], fraud detection [6, 33], as well as predicting the lethality of chemical structures in pharmaceutics [18, 39]. In all these cases it is either very expensive, dangerous, or virtually impossible to acquire (many) anomalous examples. This means that standard classification techniques cannot be applied, as there is not enough training data for each of the classes. Since there are only enough examples for one class, this problem setting is typically known as one-class classification, but for obvious reasons it can also be regarded as anomaly, or outlier, detection. The goal is simple: given sufficient training data for only the normal class, reliably detect the rare anomalies in unseen data. That is, to point the odd ones out.

Identification alone is not enough, however: explanations are also very important. This goes for classification in general, but in the one-class setup descriptions are especially important; we are deciding over rare events with possibly far-reaching consequences. A human operator will not follow advice to shut down a complex chemical installation if there is no good explanation to do so. Similarly, medical doctors are ultimately responsible for their patients, and hence will not trust a black-box telling them a patient has a rare disease if it cannot explain why this must be so.

In this chapter we give an approach for identifying anomalies in transaction data, with immediate characterisation of the why. Our approach uses the Minimum Description Length principle to decide whether an instance is drawn from the training distribution or not; examples that are very similar to the data the compressor was induced on will require only few bits to describe, while an anomaly will take many bits. By using a pattern-based compressor, thorough characterisation of its decisions is made possible. As such, our method can explain what main patterns are present/missing in a sample, identify possible observation errors, and show what changes would lead to a different decision, that is, show how strong the decision is. Furthermore, given only few anomalous examples the decision landscape can be estimated well.

We are not the first to address the one-class classification problem. However, important distinctions can be made between our approach and that of previous proposals. Here we give an overview of these differences, in Section 4.4 we discuss related work in more detail.

Most existing methods for one-class classification focus on numeric data. However, in many cases events are discrete (e.g. alarms do or do not go off, chemical sub-structures exist or not, etc.) and therefore are naturally stored in a binary or transaction database. Applying these methods on binary data is not trivial.

In classification research, high accuracy is generally the main goal. However, as pointed out above, for an expert the explanation of the decision is equally important. By their black-box nature, existing methods typically do not offer this. Our approach does, as it uses discovered patterns to describe and classify instances.

Also, by their focus on accuracy, most existing methods require the user to set a number of parameters to maximise their performance. Whereas this has obvious merit, in practice the expert will then need to fine-tune the method, while the effect

and interplays of the parameters is often unclear. Our approach does not have such parameters, making it more easily applicable.

Besides method-specific parameters, a key parameter in one-class classification is the specificity/sensitivity threshold. As there are not enough anomalies available, the decision landscape is unknown, and hence, it is difficult to set this threshold well. Our method also requires such a decision threshold. However, given only a couple of outlying examples, our method can estimate the decision landscape. As such, we provide the user with an effective way to set the decision threshold, as well as a way to see whether it is possible to identify anomalies at all.

As the compressor, here we use KRIMP [70], which describes binary data using itemsets. The high quality of these descriptions, or *code tables*, has been well established [40, 85, 86]. Alternatively, however, other compressors can be used in our framework, e.g. to apply it on other data types or with different pattern types.

Summarising, the main contributions of this chapter are two-fold. First, we provide a compression-based one-class classification method for identifying anomalies in transaction data that allows for thorough inspection of decisions. Second, we give a method that estimates the distribution of encoded lengths for anomalies very well, given only few anomalous examples. This allows experts to fine-tune the decision threshold accordingly—making our approach parameter-free for all practical purposes.

Experimentation on our method shows it provides competitive classification accuracies, reliably predicts decision landscapes, pinpoints observation errors, and most importantly, shows why decisions are made.

The remainder of the chapter is organised as follows. First, we cover the preliminaries in Section 4.2 including notation, and short introductions to MDL and one-class classification. Next, Section 4.3 covers the theory of using MDL for the one-class classification problem. Related work is discussed in Section 4.4. We experimentally evaluate our method in Section 4.5. We round up with discussion in Section 4.6 and conclude in Section 4.7.

4.2 Preliminaries

In this section we give the notation used throughout the chapter, and provide an introduction to MDL.

Notation

Throughout this chapter, as well as the following one, we consider transaction databases. Let $\mathscr{I}$ be a set of items, e.g. the products for sale in a shop. A transaction $t \in \mathscr{P}(\mathscr{I})$ is a set of items that, e.g. representing the items a customer bought in the store. A database D over $\mathscr{I}$ is then a bag of transactions, e.g. the different sale transactions on a given day. We say that a transaction $t \in D$ supports an itemset $X \subseteq \mathscr{I}$, if $X \subseteq t$. The support of X in D is the number of transactions in the database in which X occurs.

Note that any binary or categorical dataset can be trivially converted into a transaction database.

All logarithms are to base 2, and by convention $0 \log 0 = 0$.

One-Class Classification

In one-class classification, or anomaly detection, the training database D consists solely (or, overwhelmingly) of samples drawn from one distribution $\mathscr{D}_n$. The task is to correctly identify whether an unseen instance $t \notin D$ was drawn from $\mathscr{D}_n$ or not. We refer to sample being from the *normal* class if they were drawn from distribution $\mathscr{D}_n$, and to the *anomalous* class if they were drawn from any other distribution $\mathscr{D}_a$. We explicitly assume the Bayes error between $\mathscr{D}_n$ and $\mathscr{D}_a$ to be sufficiently low. That is, we assume well-separated classes—an unavoidable assumption in this setup. (Section 4.3 gives a technique to evaluate whether the assumption is valid.)

Next, we formalise this problem in terms of the Minimum Description Length principle.

MDL, a brief introduction

The Minimum Description Length principle (MDL) [24], like its close cousin MML (Minimum Message Length) [88], is a practical version of Kolmogorov Complexity [42]. All three embrace the slogan *Induction by Compression*. For MDL, this principle can be roughly described as follows.

Given a set of models $\mathscr{M}$, the best model $M \in \mathscr{M}$ is the one that minimises

$$L(M) + L(D \mid M) \,,$$

in which $L(M)$ is the length in bits of the description of M, and $L(D \mid M)$ is the length of the description of the data when encoded with model M.

This is called two-part MDL, or *crude* MDL—as opposed to *refined* MDL, where model and data are encoded together [24]. We use two-part MDL because we are specifically interested in the model: the patterns that give the best description. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases.

To use MDL, we have to define what our models $\mathscr{M}$ are, how a $M \in \mathscr{M}$ describes a database, and how all of this is encoded in bits. Note, that in MDL we are only concerned with code lengths, not actual code words.

The MDL principle implies that the optimal compressor induced on database D drawn from a distribution $\mathscr{D}$ will encode transactions drawn from this distribution more succinct than any other compressor.

More in particular, let $L(t \mid M)$ be the length, in bits, of a random transaction t, after compression with the optimal compressor M induced from database D, then

$$L(t \mid M) = -\log(\Pr(t \mid D)) \,,$$

if we assume that the patterns that encode a transaction are independent [40]. That is, under the Naïve Bayes assumption, given dataset D_1 drawn from distribution $\mathscr{D}_1$ and dataset D_2 drawn from $\mathscr{D}_2$, the MDL-optimal models M_1 and M_2 respectively induced on these datasets, and an unseen transaction t, we have the following implication

$$L(t \mid M_1) < L(t \mid M_2) \Rightarrow \Pr(t \mid D_1) > \Pr(t \mid D_2) \,.$$

Hence, it is the Bayes-optimal choice to assign t to the class of the compressor that encodes it most succinct [40].

4.3 One-Class Classification by Compression

In this section we detail the theory of using the Minimum Description Length principle for one-class classification and characterisation.

MDL for One-Class Classification

In our current setup, however, we only have sufficient training data for the normal class. That is, while we can induce M_n to model the norm, we cannot access a model M_a for the anomalies, and hence require a different way to decide whether an unseen t was drawn from $\mathscr{D}_n$ or $\mathscr{D}_a$. At the same time, however, we do know that the MDL-optimal compressor M_n will encode transactions drawn from $\mathscr{D}_n$ shorter than transactions drawn from any other distribution, including $\mathscr{D}_a$. As such, we have the following theorem.

Theorem 1. *Let t_1 and t_2 be two transactions over a set of items $\mathscr{I}$, respectively sampled from distributions $\mathscr{D}_1$ and $\mathscr{D}_2$, with $\mathscr{D}_1 \neq \mathscr{D}_2$. Further, let D be a bag of transactions sampled from $\mathscr{D}_1$, and M be the MDL-optimal compressor induced on D. Then, by the MDL principle we have*

$$L(t_1 \mid M) < L(t_2 \mid M) \Rightarrow \Pr(t_1 \mid D) > \Pr(t_2 \mid D) \,.$$

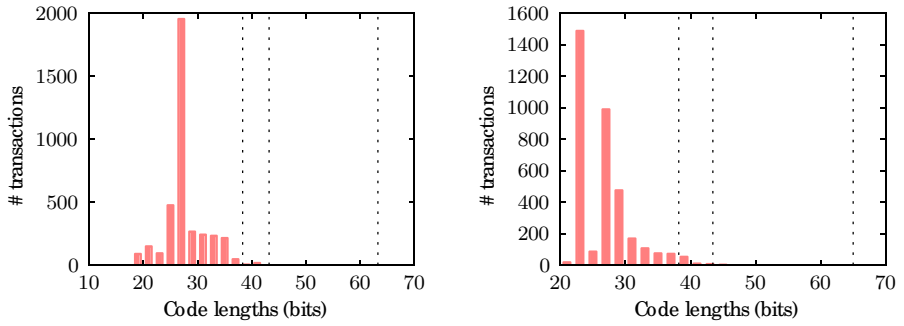
With this theorem, and under the assumption that $\mathscr{D}_n$ and $\mathscr{D}_a$ are dissimilar, we can use the encoded size of a transaction to indicate whether it was drawn from the training distribution or not. By MDL we know that if $L(t \mid M)$ is small, $\Pr(t \mid D)$ is high, and hence t was likely generated by the distribution $\mathscr{D}$ underlying D. Otherwise, if $L(t \mid M)$ is (very) large, we should regard t an anomaly, as it was likely generated by a another distribution than $\mathscr{D}$. Crucial, of course, is to determine when $L(t \mid M)$ is small enough.

The standard approach is to let the user define a cut-off value determined by the false-negative rate, i.e. the number of normal samples that will be classified as anomalies. For our setting, this would mean setting a decision threshold θ on the encoded sizes of transactions, $L(t \mid M)$, such that at least the given number of training instances are misclassified. Clearly, this approach has a number of drawbacks. First, it definitely incorrectly marks a fixed percentage of training samples as anomalies. Second, it does not take the distribution of the compressed lengths into account, and so gives an unrealistic estimate of the *real* false negative rate.

To take the distribution of encoded sizes into account, we can consider its first and second order moments. That is, its mean and standard deviation. Chebyshev's inequality, given in the theorem below, smooths the tails of the distribution and provides us a well-founded way to take the distribution into account for setting θ. It expresses that for a given random variable—in our case the compressed length, $L(t \mid M)$—the difference between an observed measurement and the sample mean is probability-wise bounded, and depends on the standard deviation.

Theorem 2 (Chebyshev's inequality [23])**.** *Let X be a random variable with expectation μ_X and standard deviation σ_X. Then for any $k \in \mathbb{R}^+$,*

$$\Pr(|X - \mu_X| \geq k\sigma_X) \leq \frac{1}{k^2} \,.$$



(a) *Class 1* (b) *Class 2*

Figure 4.1: Code length histograms for the 2 different classes of *Mushroom*, regarding separately edible or in-edible as the normal class. Shown are the compressed sizes of the transactions used to model the norm, together with the decision thresholds at false negative rates of respectively 10%, 5% and 1% estimated using Cantelli's inequality.

Note that this theorem holds in general, and can be further restricted if one takes extra assumptions into account, e.g. whether random variable X is normally distributed or not.

Given that M is the MDL-optimal compressor for a transaction database D over a set of items $\mathscr{I}$, M encodes D most succinct amongst all possible compressors. Hence we know that those transactions t over $\mathscr{I}$ with

$$L(t \mid M) < \frac{1}{|D|} \sum_{d \in D} L(d \mid M) \,,$$

will have high $\Pr(t \mid D)$. In other words, if M requires fewer bits to encode t than it requires on average for transactions from D, it is very likely that t was sampled from the same distribution as D. In order to identify anomalies, we are therefore mainly concerned with transactions that are compressed significantly worse than average. To this end, we employ Cantelli's inequality, the one-sided version of Chebyshev's inequality.

Theorem 3 (Cantelli's inequality [23])**.** *Let X be a random variable with expectation μ_X and standard deviation σ_X. Then for any $k \in \mathbb{R}^+$,*

$$\Pr(X - \mu_X \geq k\sigma_X) \leq \frac{1}{1+k^2} \,.$$

Again, like Theorem 2, this theorem holds in the general case, and we can restrict it depending on the knowledge we have on the distribution of the normal samples $\mathscr{D}_n$. Cantelli's inequality gives us a well-founded way to determine a good value for the threshold θ; instead of having to choose a pre-defined *amount* of false-negatives, we can let the user choose a confidence level instead. That is, an upper bound for the false-negative *rate* (FNR). Then, by $\theta = \mu + k\sigma$, we set the decision threshold accordingly.

Example 1. Consider Figure 4.1, depicting the histograms of the encoded sizes for the *Mushroom* database, regarding respectively the edible or in-edible mushrooms as examples for the normal class. A single histogram reflects the information the user is able to derive only looking at the encoded lengths of the normal, either edible or in-edible, examples. A standard procedure to set the decision threshold is to choose a fixed number of known false-negatives. In our setup, we opt to use the expected false-negative rate instead, for which Cantelli's gives us the correct value for the decision threshold θ. The dashed lines in Figure 4.1 show the thresholds for confidence levels of respectively 10%, 5% and 1%. The confidence level of 10%, for instance, corresponds setting θ at 3 standard deviations to the right of the average. This means the user has less than 10% chance of observing a future normal transaction that lies further than the decision threshold.

Note that using only the empirical cumulative distribution, the decision threshold would always fall inside the observed range, while by using Cantelli's inequality we are able to exceed this range: in the case above from 5% onward. Obviously, the user has no guarantees on the rate that anomalies will be classified as normal samples, i.e. the false positive rate (FPR). We will discuss that particular problem in Section 4.3.

KRIMP for One-Class Classification

In the previous subsections, we simply assumed access to the MDL-optimal compressor. Obviously, we have to make a choice for which compressor to use in practice. In this chapter we employ KRIMP, an itemset-based compressor [70], to approximate the optimal compressor for a transaction database. As such, it aims to find that set of itemsets that together describe the database best. The models KRIMP considers, *code tables*, have been shown to be of very high quality [40, 85, 86]. In Section 5.2 we will go in closer detail, here it suffices to know KRIMP describes a transaction t exactly using sets of non-overlapping itemsets, $cover(t)$, and that we denote the encoded length in bits of a transaction t by

$$L(t \mid CT) = \sum_{X \in cover(t)} L(X \mid CT) \,.$$

By running KRIMP on training database D, consisting of normal examples, we obtain an approximation of the MDL-optimal compressor for D. To employ this compressor for one-class classification, or anomaly detection, we combine the insights from Section 4.3 and 5.2. Formally, this means that given a decision threshold θ, a code table CT for database D, both over a set of items $\mathscr{I}$, for an unseen transaction t also over $\mathscr{I}$, we decide that t belongs to the distribution of D iff

$$L(t \mid CT) \leq \theta \,.$$

In other words, if the encoded length of the transaction is larger than the given threshold value, we decide it is an anomaly. We will refer to our approach as OC^3, which stands for One-Class Classification by Compression.

The MDL-optimal compressor for the normal class can obviously best be approximated when D consists solely of many samples from $\mathscr{D}_n$. In practice, however, these demands may not be met.

Firstly, D may not be very large. The smaller D is, the less well the compressor will be able to approximate the MDL-optimum. We thus especially expect good performance for large training databases. Note that MDL inherently guards against overfitting: adding too much information to a model would make it overly complex, and thus degrade compression.

Secondly, D may contain some (unidentified) samples from $\mathscr{D}_a$. However, under the assumption that $\mathscr{D}_n$ and $\mathscr{D}_a$ are well separated, the MDL-optimal compressor for a dataset D with a strong majority of samples from $\mathscr{D}_n$, and only few from $\mathscr{D}_a$, will typically encode future samples from $\mathscr{D}_n$ in fewer bits than those from $\mathscr{D}_a$. As such, the classifier will also work when the training data contains anomalies. Unless stated otherwise, in the remainder of this chapter we assume that D is sampled solely from the normal class.

Characterising Decisions

One of the main advantages of using a pattern-based compressor like KRIMP, is that we can *characterise* decisions, adopting the insights from Vreeken et al. [86].

As an example, suppose a transaction t is classified as an anomaly. That is, $L(t \mid CT) > \theta$. To inspect this decision, we can look at the itemsets by which the transaction was covered; this gives us information whether the anomaly shows patterns characteristic for the normal class. That is, the more t resembles the patterns of the normal class, the more it will be covered by long itemsets and less by singletons. On the other hand, patterns that are highly characteristic for D that are *missing* from the transaction cover are equally informative; they pinpoint where t is essentially different from the normal class.

Since code tables on average contain up to a few hundred of elements [87], this analysis can easily be done by hand. In addition, we can naturally rank these patterns on encoded size, to show the user what most characteristic, or frequently used, patterns are missing or present. As such, decisions can easily be thoroughly inspected.

Estimating the Decision Landscape

For many situations it is not unrealistic to assume that, while not abundant, *some* example anomalies are available besides the training data (e.g. less than 10). Even if these examples are not fully representative for the whole anomalous class $\mathscr{D}_a$, we can use them to make a more informed choice for the threshold parameter.

To this end, we propose to generate artificial anomalies, based on the given anomalies, to estimate the number of bits our normal-class compressor will require to encode future samples from $\mathscr{D}_a$; given this estimated distribution of encoded lengths, and the encoded lengths for the training data, we can set the decision threshold θ to maximise expected accuracy—as well as to inspect whether it is likely we will see good classification scores.

For this, we have to make one further assumption that builds on the one underlying one-class classification, i.e. that the normal and anomalous distributions are essentially different, and hence, that the MDL-optimal compressor for the normal class will badly compress anomalous samples. Now, in addition, we assume that by

slightly altering a known anomaly it will still be an anomaly. Note that if the normal and anomalous distributions are not well-separated, this assumption will not hold.

More formally, let us consider the MDL-optimal compressor M for a training database D of samples from $\mathscr{D}_n$, all over a set of items $\mathscr{I}$. Further, we have a known anomaly $t \in \mathscr{D}_a$ for which $L(t \mid M)$ is large, compared to $L(d \mid M)$ for random $d \in D$. Next, let us construct transactions t' by removing few (one, two, ...) items X from t, and replacing these with equally many items Y not in t, i.e. $t' \leftarrow (t \setminus X) \cup Y$, with $|t'| = |t|$, $Y \subseteq \mathscr{I}$ and $X \setminus t = \emptyset$. Now, the main assumption is that on average, t' is about as likely as t in $\mathscr{D}_a$, i.e. we have $\Pr(t' \mid \mathscr{D}_a) \approx \Pr(t \mid \mathscr{D}_a)$, and t' is unlikely to be drawn from $\mathscr{D}_n$, i.e. $\Pr(t' \mid \mathscr{D}_n)$ is small and $L(t' \mid M)$ relatively large. Under this assumption, $L(t' \mid M)$ gives us a good estimate of the encoded sizes of real future anomalies.

Naturally, the quality of the estimate is strongly influenced by how we swap items. One option is random change. Alternatively, we can take the compressor and the training data into account. Through these, we can identify those X and Y that will maximally change $L(t' \mid M)$; by choosing X and Y such that t' is compressed badly, we will (likely) overestimate the separation between $\mathscr{D}_n$ and $\mathscr{D}_a$, and analogously we underestimate when we minimise $L(t' \mid M)$.

We argue that in this setup the latter option is preferred. First of all, it is possible that the identified anomalies are extremes—otherwise they might not have been discovered. Second, it is not unlikely the two distributions share basic characteristics. A pattern very common in D will likely also occur in samples from $\mathscr{D}_a$; we should take this into account when sampling t's.

Given some prototype anomalies, we generate new samples according to the following distribution. First, we uniformly choose a transaction t among the given prototype anomalies. Next, from t we select an item i to remove, using the following exponential distribution,

$$\Pr(i) = \frac{2^{-1/l(i)}}{\sum_{j \in t} 2^{-1/l(j)}} ,$$

where, $l(i) = \frac{L(Z|CT)}{|Z|}$ and $i \in Z \in \mathit{cover}(t)$. By this choice, we prefer to remove those items that require the most bits to be described—that is, those that fit the patterns from the normal class least. To complete the swap, we choose an item from $\mathscr{I} \setminus t$ to add to t. (Note that if the original dataset is categorical, it only makes sense to swap to items corresponding to the same category.) We choose the item j to swap to according to the following distribution, similar to how Vreeken and Siebes [85] imputed missing values,

$$\Pr(j) = \frac{2^{-L(t_j|CT)}}{\sum_{k \in \mathscr{I} \setminus (t \setminus i)} 2^{-L(t_k|CT)}} ,$$

with $t_j = (t \setminus \{i\}) \cup \{j\}$. This distribution generates transactions t' with preference to short encoding.

To estimate the expected false positive rate, we generate a large number of samples and calculate mean and standard deviation. One can use Cantelli's inequality, or assume the encoded lengths of the anomalies to follow a normal distribution. Then, one can update θ by taking both FPR and FNR into account, e.g. choose the intersection between the two distributions.

Measuring Decision Certainty

Item swapping is also useful to show how a transaction t needs to be modified in order to change the classification verdict. Or, the other way around, to show what items are most important with regard to the decision of t. However, we can go one step further, and look at the certainty of a decision by considering the encoded lengths of altered transactions. The rationale is that the more we need to change t to let its encoded length reach below the decision threshold, the more likely it is this example is indeed an anomaly. Alternatively, for a sample with an observation error, a small change may be enough to allow for a correct decision.

So, the goal is, given a transaction t, to maximally reduce the encoded size $L(t \mid CT)$ with a limited number of changes δ. In general, transactions may have different cardinality, so up to δ elements can be added—in categorical databases transactions are of the same size and up to δ items need to be swapped. Clearly, with $\binom{|\mathscr{I} \setminus t|}{\delta} \times \binom{|t|}{\delta}$ possible altered transactions, solving this problem exhaustively quickly becomes infeasible for larger δ and $\mathscr{I}$. However, in our setup we can exploit the information in the code table to guide us in choosing those swaps that will lead to a short encoding.

The idea is to cover the transaction t using the most specific elements, i.e., the itemsets $X \in CT$ with highest cardinality $|X|$, while tolerating up to δ missing items. The reason to choose the most specific elements is that we cover the most singletons by one itemset, and therewith replace many (on average long) codes by just one (typically short) code. Note that, alternatively, one could attempt to greedily cover with the elements with the shortest codes, or even minimise the encoded length by dynamic programming.

We present the pseudo-code for finding a δ-fault-tolerant cover for a transaction t and a code table CT as Algorithm 2. In order to calculate the resulting encoded length of t, we simply use the code lengths in the code table. In the algorithm, while covering t, we keep track of the number of swaps made so far, denoted as ϵ. Once the number of uncovered items in t is smaller or equal than ϵ (line 2) we can stop: the items remaining in t are the items we have to remove, the items $S \setminus t$ are the ones we have to add. We only use Algorithm 2 as a step during analysis of decisions; it would require non-trivial adaptations to KRIMP to let it consider missing items when constructing the optimal code table, and our focus here is clearly not to construct a new compressor.

4.4 Related Work

Much of the existing work on one-class classification targets record data constructed with numerical attributes. For a general overview, we refer to [49, 77]. Very few of these studies are applicable to transaction data, as many of the these methods rely on density estimations (e.g. Parzen-windows or mixture of Gaussians) to model the norm. Two state-of-the-art algorithms that, by respectively using the appropriate kernel or distance measure, are applicable to binary data are Support Vector Data Description (SVDD) [79], or one-class SVM [66], and Nearest Neighbour Data Description (NNDD) [77].

He et al. [28] study outlier detection for transaction databases. The authors assume that transactions having less frequent itemsets are more likely to be outliers.

Algorithm 2 FAULT-TOLERANT COVER

Input: Transaction $t \in D$ and code table CT, with CT and D over a set of items $\mathscr{I}$. Maximum number of faults δ, and current number of faults ϵ.

Output: Fault-tolerant cover of t using the most specific elements of CT, tolerating at most δ faults.

1. $S \leftarrow$ smallest X of CT in **Standard Cover Order** with $|X \setminus t| \leq \delta$, $X \in t$ if $|X| = 1$
2. **if** $|t \setminus S| \leq \epsilon$ **then**
3. $\quad Res \leftarrow \{S\}$
4. **else**
5. $\quad Res \leftarrow \{S\} \cup$ **Fault-Tolerant Cover**$(t \setminus S, CT, \delta - |S \setminus t|, \epsilon + |S \setminus t|)$
6. **end if**
7. **return** Res

Narita et al. [56], on the other hand, use information of association rules with high confidence for the outlier degree calculation. Both these approaches were formulated to detect outliers in a database, but can also be used for single-class problems. While both methods use patterns and share the transparency of our approach, their performance is very parameter-sensitive. For the former, the authors restrict themselves to the top-k frequent itemsets. In the latter, besides a minimum support threshold, the user also needs to specify minimum confidence. Both papers notice large changes in accuracy depending on the parameter settings, but postpone insight in how to set these optimally to future work.

We are not the first to employ the MDL principle for one-class classification. However, to the best of our knowledge, we are the first to apply it in a binary/transaction database setting. Bratko et al. [8] and Nisenson et al. [58] consider streams of character data (e.g. text, streams of bytes or keyboards events, etc.). In these approaches, the compressor is immaterial; that is, well-known universal compression algorithms, such as gzip, are used, which do not allow for inspection. The algorithms compress common shared (sub)strings of characters that occur often together in the streams. In transaction databases one is not interested in sequences of items, as items are unordered.

We use KRIMP, introduced by Siebes et al. [70], to build a compression model relying on the (frequent) itemsets to encode transactions. Van Leeuwen et al. show that these models are able to compete with the state-of-the-art multi-class classifiers [40]. Moreover, Vreeken et al. demonstrate how to identify and characterise differences among databases or different classes in a database based on the information of encoded lengths [86]. Alternatively, one could use PACK [76], LESS [29], or any other suited transaction data compressor, in our framework.

4.5 Experiments

In this section we experimentally evaluate our approach. First, we discuss the experimental setup, then investigate classification accuracy. Next, we show how classification decisions can be characterised, and observation errors in transactions can be detected. Finally, we estimate the distribution of encoded lengths for anomalies to improve classification results.

Table 4.1: Statistics of the datasets used in the experiments.

				KRIMP	
Dataset	$\|D\|$	$\|\mathscr{I}\|$	%1s	$\|\mathscr{K}\|$	*minsup*
Adult	48842	95	15.3	2	50
Anneal	898	66	20.2	5	1
Breast	699	14	62.4	2	1
Chess (k-k)	3196	75	49.3	2	400
Chess (kr-k)	28056	58	12.1	18	1
Connect-4	67557	129	33.3	3	1
Heart	303	46	28.0	4	1
Iris	150	16	26.3	3	1
Led7	3200	14	33.3	10	1
MCADD	32916	195	11.1	1	128
Mushroom	8124	117	19.3	2	1
Nursery	12960	27	28.1	4	1
Pageblocks	5473	39	25.0	5	1
Pen Digits	10992	76	19.8	10	10
Pima	768	36	23.7	2	1
Tic–Tac–Toe	958	27	34.5	2	1
Typist	533	40	25.7	10	1
Wine	178	65	20.6	3	1

Given are, per dataset, the number of rows, the number of distinct items, density (in percentage of 1s), the number of classes and the *minsup* thresholds for the KRIMP-compressor.

Experimental Setup

For the experimental validation of our method we use a subset of publically available discretised datasets from the LUCS-KDD repository [15]. In addition to these datasets, shown in Table 4.1, we also consider the *Typist* recognition problem provided by Hempstalk et al. [30], discretised and normalised using the LUCS-KDD DN software [15].

We turn this selection of multi-class classification datasets into several one-class classification problems. For each dataset, one class at a time, we consider a particular class $K \in \mathscr{K}$ as the normal class and the samples from the remaining classes $\mathscr{K} \setminus K$ as anomalies. All results reported in this section are 10-fold cross-validated.

To obtain the KRIMP code table CT_K for a dataset D_K, we use (closed) frequent itemsets in D_K mined with *minsup* set as low as practically feasible. The actual values for *minsup* are depicted in Table 4.1.

One-Class Classification

We compare our method to two state-of-the-art [30, 79] one-class classifiers: Support Vector Data Description (SVDD) [66, 79] and Nearest Neighbour Data Description (NNDD) [77], both of which are publically available in DDtools [78].

For the kernel in SVDD, or one-class SVM, we use the polynomial kernel, and optimise the degree parameter d for high accuracy. While more generally employed in one-class classification, for binary data the RBF kernel leads to worse results. One other advantage of using polynomial kernels with binary data is the close relatedness to itemsets: the attributes in the feature space induced by this kernel indicate the presence of itemsets up to length d.

For NNDD we use the Hamming distance. To determine the number of neighbouring points that are used to decide the classification, parameter k, we optimise the log-likelihood of the leave-one-out density estimation [78].

To compare the algorithms, we use the AUC, i.e. area under the ROC curve, as it is independent of actual decision thresholds. Table 4.2 shows the AUC scores averaged over 10-folds and each of the classes of each dataset. We see, and it is confirmed by pairwise Student's t-tests at α-level 5%, that, in general, the performance of OC^3 is on par with that of both other algorithms, and that SVDD clearly outperform NNDD. Especially for datasets with high numbers of transactions, e.g. *Chess* (*kr-k*), *Connect-4*, *Mushroom*, and *Pen Digits*, OC^3 provides very good performance, while for data with very small classes, such as *Iris*, *Tic–Tac–Toe* and *Wine*, it does not improve over the competing methods. This is expected, as MDL requires sufficient samples to properly estimate the training distribution. In conclusion, OC^3 performs on par with the state of the art, and can improve over it for large databases.

The sub-par performance of all classifiers on *Pageblocks* and *Pima*, can be explained: these datasets contain large numbers of identical transactions that only differ on class-label, making errors unavoidable.

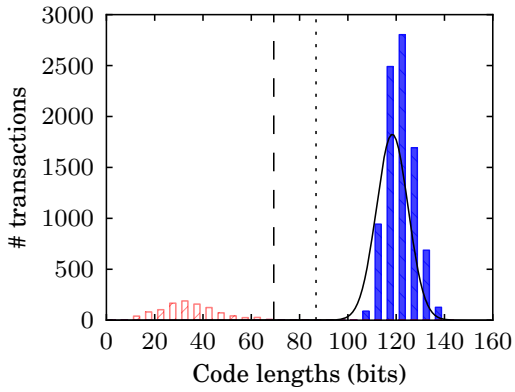
In our setup, classification depends on code length distributions for the different classes. Figure 4.2a and 4.2b show the histograms on the training instances of two classes from the *Pen Digits* dataset. One can notice that the normal transactions (shown in hatched red) are better compressed than the anomalies, i.e. the other classes, (filled blue) which is in accordance with the MDL assumption. The quality of the classification result is determined by the amount of overlap. Following, the more the two distributions are apart, the better the results are. For the sub-par performing datasets in Table 4.2, the histograms overlap virtually completely, and hence the separation-assumption is not met.

If memory or time constraints do not allow us to mine for frequent itemsets at low *minsup*, e.g. for the dense *Chess* (*k-k*) dataset, some characteristic patterns might be missing from the code tables, in turn leading to sub-par compression and performance. Clearly, at further expense, mining at lower *minsup* would provide better compression, which in turn should provide better performance on these datasets.

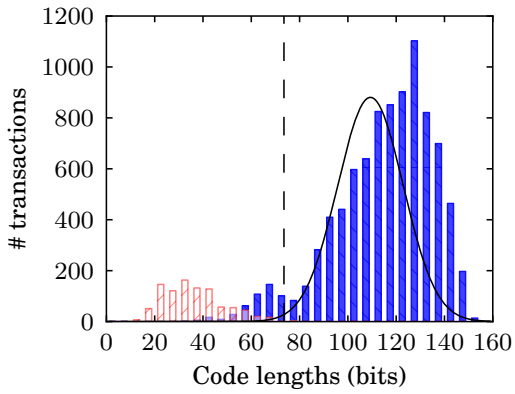
Table 4.2: AUC scores for the benchmark datasets.

Dataset	OC^3	SVDD	NNDD
Adult	68.63 ± 3.71	72.86 ± 6.68	*65.64 ± 5.84*
Anneal	95.58 ± 0.62	93.62 ± 9.95	97.32 ± 2.36
Breast	87.12 ± 12.76	**96.47 ± 1.19**	72.77 ± 33.04
Chess (k-k)	68.57 ± 0.58	65.62 ± 7.89	**82.27 ± 1.69**
Chess (kr-k)	**94.89 ± 5.18**	86.46 ± 8.71	80.38 ± 9.79
Connect-4	**73.73 ± 6.47**	55.14 ± 6.98	62.22 ± 5.52
Heart	65.13 ± 9.99	69.06 ± 12.15	66.18 ± 7.85
Iris	*93.39 ± 3.84*	98.20 ± 2.08	95.96 ± 4.60
Led7	91.45 ± 3.50	93.41 ± 3.45	79.43 ± 7.26
Mushroom	**100.00 ± 0.00**	97.69 ± 2.89	99.92 ± 0.07
Nursery	98.43 ± 1.68	98.68 ± 1.68	84.54 ± 7.16
Pageblocks	52.59 ± 23.87	56.79 ± 13.91	51.13 ± 13.07
Pen Digits	98.25 ± 0.89	98.98 ± 0.80	98.32 ± 1.00
Pima	50.94 ± 28.93	65.32 ± 9.66	*50.63 ± 12.81*
Tic–Tac–Toe	89.90 ± 7.15	98.74 ± 1.78	97.89 ± 1.22
Typist	87.81 ± 6.93	92.30 ± 6.35	87.84 ± 7.73
Wine	91.37 ± 0.66	97.85 ± 1.18	95.51 ± 5.26
average	82.81 ± 8.22	84.54 ± 4.20	*80.47 ± 7.64*

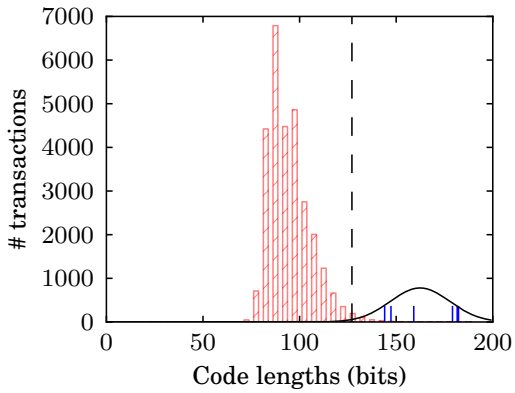
Shown are, per dataset, mean and standard deviation of the average AUC score over the classes. The KRIMP-compressor in OC^3 ran using the all closed itemsets above *minsup* values in Table 4.1 as candidates. Highlighted are the comparative results using pairwise Student's t-tests at α-level 5%: bold and underlined values indicate AUC scores significantly better or worse than the other two, while italics signal scores that only differ significantly from those of SVDD.



(a) *Pen Digits* - Class 1

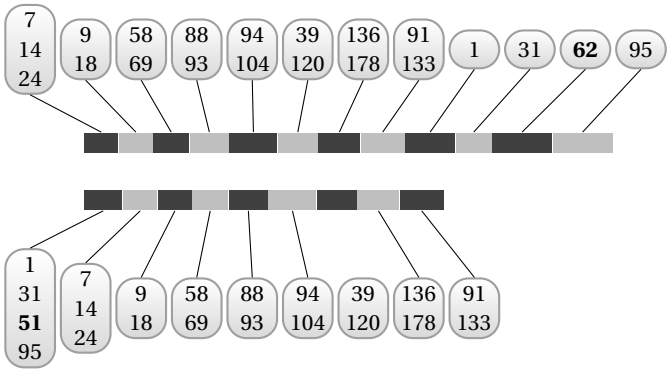


(b) *Pen Digits* - Class 10

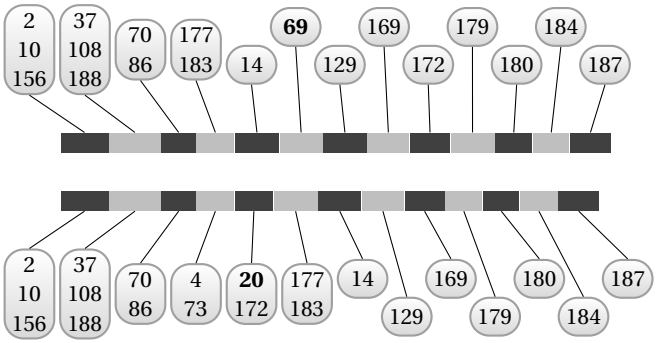


(c) *MCADD* - Healthy Class

Figure 4.2: Code length histograms for *MCADD* and *Pen Digits*. Shown are the compressed sizes of transactions of the normal ones (in hatched red) and the anomalies (in filled blue). The solid black line depicts our estimate, using 4 counterexamples, of the encoded lengths for the anomalies. The dashed line represents the decision threshold bounding the FPR at 10%, while the dotted line, if present, shows the decision threshold after updating.



(a)



(b)

Figure 4.3: Example transactions from *MCADD*: observation error (top a), true anomaly (top b) and corrections (bottom) suggested by Algorithm 2 ($\delta = 1$). The rounded boxes visualise the itemsets that are used to cover the transaction; each itemset is linked to its code. Width of the bar represents the length of the code. Swapped items are displayed in boldface. Clearly, the observation error is correctly identified, as its encoded length can be decreased much with only swap, whereas the encoded length of the anomaly cannot be improved much.

Inspection and Characterisation

One of the key strengths of our approach is that we can analyse and explain the classification of transactions. Here, we investigate how well the approach Section 4.3 outlines works in practice. Note that black-box methods like SVDD and NNDD do not allow for characterisation.

To demonstrate the usability of our approach in a real problem setting, we perform a case study on real MCADD data, described in detail in Section 4.5, obtained from the Antwerp University Hospital. Medium-Chain Acyl-coenzyme A Dehydrogenase Deficiency (MCADD) [5] is a deficiency newborn babies are screened for during a Guthrie test on a heel prick blood sample. This recessive metabolic disease affects about one in 10 000 people while around one in 65 is a carrier of the responsible mutated gene. If left undiagnosed, this rare disease is fatal in 20 to 25% of the cases and many survivors are left with severe brain damage after a severe crisis.

Figure 4.3, which shows the covers of 4 transactions from the *MCADD* dataset, serves as our running example. A typical cover of a transaction from the normal class is shown in Figure 4.3a (bottom): one or more larger itemsets possibly complemented with some singletons. Also note that the lengths of the individual codes, denoted by the alternating light and dark grey bars, are short. This is in strong contrast with the covers of the other transactions, which resemble typical covers for anomalous transactions, where mostly singletons are used. Also, the code lengths of itemsets in the cover of an anomalous transaction are typically long.

The 'anomaly' transaction at the top of Figure 4.3a was artificially constructed by taking a transaction from the normal class. If we use Algorithm 2, with δ, the number of mistakes allowed, set to one, we exactly recover the true normal transaction (bottom of Figure 4.3a). This shows that we are able to detect and correct observation errors in future samples. Or, the other way around, if many swaps are needed for the decision to change, this gives a plausible explanation why the transaction is identified as an anomaly.

The top transaction in Figure 4.3b is a true anomaly. We first of all observe that the encoded size of the transaction is large. Next, as the items are labeled descending on their support, we see that more than halve of the items belong to the 20% least frequent. Moreover, we note that by applying Algorithm 2 the gains in encoded length are negligible (see bottom of Figure 4.3b for one example). This trend holds in practice, and strengthens confidence in the assumption made in Section 4.3, that small variations of anomalies remain 'anomalous'. However, as shown above, perturbing normal samples can cause large variation in encoded size as larger patterns fall apart into singletons.

Estimating Anomaly Distributions

Next, we investigate how well we can estimate the distribution of encoded lengths for anomalies, by generating samples from a limited number of anomalies, as outlined in Section 4.3. For both the *MCADD* and *Pen Digits* dataset, we generated 10 000 samples based on 4 randomly selected anomalies. As shown in Figure 4.2, the so-derived normal distributions, based on the sample mean and standard variation, approximate the true anomaly distributions closely. Note that, as intended, the estimate is slightly conservative.

Alternatively, if we uniformly swap items, the patterns shared by both anomalies and the expected ones are more likely to be destructed. Experiments show this provides overly optimistic estimates of the separation of the distributions. Further, in many situations, it is sensible to estimate pessimistically. That is, closer to the normal samples. For example, if MCADD is left undiagnosed, it is fatal in 20% to 25% of the cases, and survivors are typically left with severe brain damage.

We will now use illustrate through Figure 4.2a and 4.2b how a user can use this information to update the decision threshold θ. Initially, the user only has information from the normal class, denoted by the hatched red histograms. By using Cantelli's inequality, the decision threshold can be set to allow up to 10% false negatives (dashed line). After estimating the distribution for the anomalies, we notice this initial choice fits perfect for class 10 in Figure 4.2b. However, the decision threshold for class 1 in Figure 4.2a is too low (5% FNR, 0.2% FPR). If we update the decision threshold (dotted line) to counterbalance both the estimated false negative and false positive rate using Cantelli's inequality, we observe that the false negative and false positive rates on the hold-out validation set are more in balance: 1.8% FNR and 1.5% FPR. So, by using information derived from the artificial anomalies, the user can update the threshold and improve classification results.

Case study: MCADD

In our study, the dataset contains controls versus MCADD, with respectively 32 916 normal ones and only 8 anomalies. The instances are represented by a set of 21 features: 12 different acylcarnitine concentrations measured by tandem mass spectrometry (TMS), together with 4 of their calculated ratios and 5 other biochemical parameters. We applied k-means clustering with a maximum of 10 clusters per feature to discretise the data resulting in 195 different items. We run KRIMP using a minimum support of 15, which corresponds to a relative *minsup* of 0.05%.

Repeated experiments using 10-fold cross-validation show that all 8 anomalous cases are ranked among the top-15 largest encoded transactions. Besides, we notice that the obtained performance indicators (100% sensitivity, 99.9% specificity and a predictive value of 53.3%) correspond with the state-of-the-art results [5, 31] on this problem. Moreover, analysing the anomalous cases by manually inspecting the patterns in the code table and covers, reveals that particular combinations of values for acylcarnitines C2, C8 and C10 together with particular following ratios $\frac{C8}{C2}$, $\frac{C8}{C10}$ and $\frac{C8}{C12}$ were grouped together in the covers of the abnormal cases. Exactly these combination of variables are commonly used in diagnostic criteria by experts and were also discovered in previous in-depth studies [5, 31].

The largest outlying samples stand out as a rare combination of other acetylcarnitine values. Although these samples are not MCADD cases, they are very different from the general population and are therefore abnormal by definition.

4.6 Discussion

The experiments in the previous section demonstrate that our method works: transactions that are succinctly compressed by patterns from the normal class are indeed highly likely to belong to that class. The obtained AUC scores are on par with the state-of-the-art one-class classifiers for binary data, and especially good for large datasets.

In practice, a user lacks an overall performance measure to reliably specify the specificity/sensitivity threshold, as anomalous examples are rare in one-class classification. Consequently, the ability to inspect classification decisions is important. In contrast to existing approaches, our pattern-based method provides the opportunity to analyse decisions in detail.

Examples in the experiments show possible use cases. First, we are able to explain why a transaction is classified as such. Transactions that are covered with itemsets that are highly characteristic for the normal class are likely normal as well, while those transactions that do not exhibit such key patterns (and thus encoded almost solely by singletons) can be considered as anomalies. Next, our approach is able to detect, and correct, observation errors in test samples.

Furthermore, if some anomalies are available, we propose to approximate the encoding distributions of the anomalies. By using this information, a user is able to make a more informed decision when setting the decision threshold. Here, we choose to generate samples conservatively. Visualising the approximated distribution of encoded lengths for anomalies shows whether one-class classification, based on compressed lengths, is actually possible.

A case study on the MCADD dataset shows that true anomalies are correctly identified. Different from the setup explored here, where unseen transactions are considered as potential anomalies, one could also be interested in detecting anomalies *in* the dataset at hand. The method discussed in this chapter may well provide a solution for this problem, that is, pointing out the most likely outlying items and transactions by compressed size. Related, as a future work, we are investigating a rigorous approach for cleaning data using MDL.

Although in this chapter we focus on binary data, the methods we present can be generalised as a generic approach using MDL. That is, as long as a suited compressor is employed, the theory will not differ for other data types. Variants of KRIMP have already been proposed for sequences, trees, and graphs [4].

4.7 Conclusion

In this chapter we introduced a novel approach to anomaly detection, or one-class classification, for binary or transaction data. In this setting little or no examples are available for the class that we want to detect, but an abundance of normal samples exists. Our method is based on the Minimum Description Length principle, and decides by the number of bits required to encode an example using a compressor trained on samples of the normal situation. If the number of bits is much larger than expected, we decide the example is an anomaly. Experiments show that this approach provides accuracy on par with the state of the art.

Most important, though, is that it holds three advantages over existing methods. First, by relying on pattern-based compression, our method allows for detailed inspection and characterisation of decisions, both by showing which patterns were recognised in the example, as well as by checking whether small changes affect the decision. Second, we show that given a few anomalous examples, our method can reliably estimate the decision landscape. Thereby, it can predict whether the normal class can be detected at all, and allows the user to make an informed choice for the decision threshold. Third, given this estimate the method is parameter-free.

CHAPTER 5

Directly Mining Descriptive Patterns

Mining small, useful, and high-quality sets of patterns has recently become an important topic in data mining. The standard approach is to first mine many candidates, and then to select a good subset. However, the pattern explosion generates such enormous amounts of candidates that by post-processing it is virtually impossible to analyse dense or large databases in any detail.

We introduce SLIM*, an any-time algorithm for mining high-quality sets of itemsets directly from data. We use MDL to identify the best set of itemsets as that set that describes the data best. To approximate this optimum, we iteratively use the current solution to determine what itemset would provide most gain—estimating quality using an accurate heuristic. Without requiring a pre-mined candidate collection,* SLIM *is parameter-free in both theory and practice.*

Experiments show we mine high-quality pattern sets; while evaluating orders-of-magnitude fewer candidates than our closest competitor, KRIMP*, we obtain much better compression ratios—closely approximating the locally-optimal strategy. Classification experiments independently verify we characterise data very well.*

This chapter is based on work published as [74]:
K. Smets and J. Vreeken. SLIM: Directly mining descriptive patterns. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA*, 2012.

5.1 Introduction

Pattern mining is one of the central topics in data mining, and is focused on discovering interesting local structure in data. Starting from frequent sets and association rules [2], one of the key goals in pattern mining has been completeness: discovering all patterns that satisfy certain conditions. In a way, this is a useful goal, as we know that all returned patterns meet the requirements—and as such, are potentially interesting.

Completeness, however, also has its drawbacks. Typically, there exist extremely many patterns fulfilling the constraints, many of which convey the same information. Mining all patterns hence leads to prohibitively large and strongly redundant results, which are in turn difficult to use or interpret. To address this, researchers have recently instead focused on discovering small and useful, high-quality *sets of patterns* [37, 52, 87].

We identify the best set of patterns as those patterns that together describe the data best. That is, by the Minimum Description Length (MDL) principle [24], we are after that set of patterns that provides the optimal lossless compression of the data. By definition, this set has many desirable properties: it is non-redundant, not overly simple, nor complex for the data—as otherwise, bits could have been saved. The main question is, how do we mine this optimal result?

In this chapter, we present SLIM, an efficient heuristic for directly mining high-quality data descriptions on transaction data. SLIM is a fast, one-phase, any-time alternative to KRIMP [70]. Besides obtaining better compression, it can handle large and dense datasets, and is parameter-free in both theory and practice.

The MDL approach to pattern mining was pioneered by Siebes et al. [70, 87], who gave the KRIMP algorithm to approximate the optimal set of itemsets, or, *code table*. Subsequent research showed these sets of patterns to be very useful. Besides characterising the distribution of the data very well, they have been shown to provide high performance in a wide range of tasks, including difference measurement [86], clustering [41], and anomaly detection [73].

Like many pattern set mining approaches [10, 36, 69], KRIMP follows a straightforward two-phase approach to mining code tables. First, it mines a collection of frequent itemsets. Second, it considers these candidates in a static order, accepting a pattern if it improves compression. This simplicity has some drawbacks.

First of all, mining candidates is expensive. As more candidates correspond to a larger search space, lower support thresholds correspond to better final results. Often, however, it is difficult to keep the number of candidates feasible—for large and dense databases especially, a small drop of the threshold can lead to an enormous increase in patterns. Moreover, as most candidates will be rejected, this step is quite wasteful.

Second, by considering candidates only once, and in a fixed order, KRIMP sometimes rejects candidates that it could have used later on. A more powerful strategy would be to iteratively select the best addition out of all candidates—which naively, however, quickly becomes infeasible for larger databases or candidate collections.

With SLIM, we address these issues, and give an efficient any-time algorithm for mining descriptive pattern sets *directly* from data. We greedily construct pattern sets in a bottom-up fashion, iteratively joining co-occurring patterns such that compression is maximised. To reduce computation and database scans, it employs a

simple yet accurate heuristic to estimate the gain or cost of introducing a candidate. Importantly, SLIM is parameter-free in both theory and practice.

Experiments show we discover high-quality pattern sets. The SLIM code tables attain very high compression ratios; in particular on large and dense datasets, we obtain tens of percentages better compression than KRIMP, while considering orders-of-magnitude fewer candidates. Classification experiments show high accuracy, verifying that we characterise the data well. Convergence plots show SLIM closely approximates the powerful greedy approach of selecting the best candidate out of all candidates; all while being much more efficient.

The remainder of this chapter is organised as follows. First, we cover the preliminaries in Section 5.2 including short introductions to code tables and KRIMP. Next, in Section 5.3 we discuss how to mine and identify good candidate patterns. In Section 5.4 we introduce the SLIM algorithm for mining high quality code tables directly from data. Section 5.5 discusses related work, and we experimentally evaluate our method in Section 5.6. Finally, we round up with discussion in Section 5.7 and conclude in Section 5.8.

5.2 Preliminaries

In this section we provide an introduction to code tables and the KRIMP algorithm, and refer to Section 4.2 for the notation used throughout the chapter and a brief introduction to MDL.

Code tables

Recall from Section 4.2 that in order to use MDL, we have to define what our models are, how a model describes a database, and how we encode these in bits.

As our itemset-based models we use code tables [70, 87]. A code table is a simple dictionary: a two-column table with itemsets on the left-hand side, and corresponding codes on its right-hand side. The itemsets in the code table are ordered first descending on cardinality, second descending on support, and third lexicographically. We refer to this as the **Standard Cover Order**.

The actual codes on the right-hand side are of no importance: their lengths are. To explain how these lengths are computed, the coding algorithm needs to be introduced. A transaction t is encoded by searching for the first itemset X in the code table for which $X \subseteq t$. The code for X becomes part of the encoding of t. If $t \setminus X \neq \emptyset$, the algorithm continues to encode $t \setminus X$. Since it is insisted that each code table contains at least all single items, this algorithm gives a unique encoding to each (possible) transaction over $\mathscr{I}$.

The set of itemsets used to encode a transaction is called its *cover*. Note that the coding algorithm implies that a cover consists of non-overlapping itemsets. The length of the code of an itemset in a code table CT depends on the database at hand; the more often a code is used, the shorter it should be. To this end, we use an optimal prefix code. To compute the code lengths, we have to cover every transaction in the database.

The *usage* of an itemset $X \in CT$ is the number of transactions $t \in D$ which have X in their cover. The relative usage of $X \in CT$ is the probability that X is used in the

encoding of an arbitrary $t \in D$. For optimal compression of D, the higher $\Pr(X \mid D)$, the shorter its code should be. In fact, from information theory [16], we have the Shannon entropy, which gives us the length of the optimal prefix code for X as

$$L(X \mid D) = -\log \Pr(X \mid D) \,,$$

where

$$\Pr(X \mid D) = \frac{usage(X)}{\sum\limits_{Y \in CT} usage(Y)} \,.$$

The length of the encoding of transaction is now simply the sum of the code lengths of the itemsets in its cover,

$$L(t \mid CT) = \sum_{X \in cover(t)} L(X \mid CT) \,.$$

The size of the encoded database is then the sum of the sizes of the encoded transactions,

$$L(D \mid CT) = \sum_{t \in D} L(t \mid CT) \,.$$

To find the optimal code table using MDL, we need to take both the compressed size of the database and the size of the code table into account. For the size of the code table, we only consider those itemsets that have a non-zero usage. The size of the right-hand side column is obvious; it is simply the sum of all the different code lengths. For the size of the left-hand side column, note that the simplest valid code table consists only of the single items. This code table we refer to as the **Standard Code Table**, or ST. We encode the itemsets in the left-hand side column using the codes of ST. This allows us to decode up to the names of items. The encoded size of the code table is then given by

$$L(CT \mid D) = \sum_{\substack{X \in CT \\ usage(X) \neq 0}} L(X \mid ST) + L(X \mid CT) \,.$$

We define the optimal set of itemsets as the one whose associated code table minimises the total encoded size

$$L(CT, D) = L(CT \mid D) + L(D \mid CT) \,.$$

More formally, we define the problem as follows.

Minimal Coding Set Problem *Let $\mathscr{I}$ be a set of items and let D be a dataset over $\mathscr{I}$,* cover *a cover algorithm, and $\mathscr{F}$ a collection of candidate patterns $\mathscr{F} \subseteq \mathscr{P}(\mathscr{I})$. Find the smallest set of patterns $\mathscr{S} \subseteq \mathscr{F}$ such that for the corresponding code table CT the total compressed size, $L(CT, D)$, is minimal.*

Using our cover algorithm only patterns occurring in D can be used in describing the data. As such, $\mathscr{P}(\mathscr{I})$ is a clear overestimate for what candidates the optimal CT can consist of. For reasons of efficiency, and without loss of generality, we can hence limit $\mathscr{F}$ to the collection of all itemsets in D with a support of at least 1.

Even for small $\mathscr{F}$, however, the search space of all possible code tables is rather large—and moreover, it does not exhibit structure nor monotonicity we can use to efficiently find the optimal code table [87]. Hence, we resort to heuristics.

Algorithm 3 The KRIMP Algorithm [87]

Input: A transaction database D and a candidate set $\mathscr{F}$, both over a set of items $\mathscr{I}$
Output: A heuristic solution to the Minimal Coding Set Problem, code table CT

1. $CT \leftarrow$ **Standard Code Table**(D)
2. **for** $F \in \mathscr{F}$ in **Standard Candidate Order do**
3. $\quad CT_c \leftarrow (CT \oplus F)$ in **Standard Cover Order**
4. $\quad$ **if** $L(D, CT_c) < L(D, CT)$ **then**
5. $\quad\quad CT \leftarrow$ *post-prune*(CT_c)
6. $\quad$ **end if**
7. **end for**
8. **return** CT

Introducing KRIMP

The KRIMP algorithm was introduced by Siebes et al. [70, 87] as a straightforward approach for mining good approximations of the optimal code table. Subsequent research showed these code tables to indeed be of high quality, and useful for a wide range of data mining tasks [41, 73, 86, 87].

The pseudo-code of KRIMP is given as Algorithm 3. It starts with the singleton code table (line 1), and a candidate collection $\mathscr{F}$ of frequent itemsets up to a given *minsup*. The candidates are ordered first descending on support, second descending on itemset length and third lexicographically. Each candidate F is considered in turn by inserting it in CT (3), denoted by $CT \oplus F$, and calculating the new total compressed size (4). It is only accepted if compression improves (4). If accepted, all elements $X \in CT$ are reconsidered wrt. their contribution to compression (5).

The *minsup* parameter is used to control the number of candidates: the lower *minsup*, the more candidates, the larger the search space, and hence the better the approximation of the optimal code table. As by MDL we are after the optimal compressor, *minsup* should be set as low as practically feasible. Hence, technically *minsup* is not a true parameter.

In practice, however, keeping the number of itemsets feasible is difficult. Especially for dense or large databases, minute decreases in threshold give rise to enormous increases in patterns. Mining, sorting, and storing these in large numbers takes non-trivial time and effort, even before KRIMP can begin selecting.

Its robust empirical results aside, KRIMP is a rather rough greedy heuristic: it considers each candidate only once, in a static order, raising the question how good its approximations really are.

A standard approach to hard optimisation problems, with provable approximation bounds in case of sub-modular set functions [57], is to locally optimise the target function. For KRIMP, local optimisation translates to iteratively finding the $F \in \mathscr{F}$ that maximally increases compression. By considering $\mathscr{F}$ quadratically instead of linearly, however, the running time quickly grows out of hand. In Section 5.6 we will refer to this variant as KRAMP. While there are no results on sub-modularity for code tables, and hence no approximation bounds, we can use it as a gold-standard to compare to.

5.3 Identifying good candidates

Instead of filtering pre-mined candidates, we rather mine code tables directly from data. To do so, we need to be able to identify good candidates on the fly.

Covering Data

Intuitively, SETCOVER seems like a suited approach for approximating the optimal code table. Its goal is to find the smallest set of itemsets that cover all 1s in the data. This problem is NP-hard, but good approximation bounds exists for the greedy approach. That approach, also known as Tiling [22], iteratively finds that itemset (or, tile) by which the most uncovered 1s in the data are covered.

In practice, however, Tiling does not mine good code tables. Whereas KRIMP refines its cover by replacing general patterns with more specific ones, Tiling only focuses on covering data; quickly leading to the selection of overly general (individual items), or overly specific itemsets (complete transactions), that do not contribute towards good compression.

Interestingly, though, we do see that the first few discovered tiles typically do compress well. This is especially interesting, as we further note that both methods initially regard the data similarly: at first, every 1 in the data corresponds to a separate code, and hence, covering many 1s with one tile means replacing many codes with one code. This suggests finding tiles may be worthwhile, if we consider the right search space.

(Note that as in our problem itemset costs develop non-monotonically with changes of CT, the weighted variant of SETCOVER does not apply trivially.)

Covering Codes

Whereas Tiling only regards the uncovered part of the 0/1 data matrix, in our problem we always consider complete covers. That is, the introduction of a new code table element X into CT induces a different covering of the data—for which we determine its quality by $L(CT, D)$.

We can regard the cover of a database D by a code table CT as follows. Let $\mathscr{C}$ be the $|D|$-by-$|CT|$ binary matrix, where the rows correspond to transactions $t \in D$ and the columns to elements $X \in CT$. The cell values are 1 when $X \in \mathit{cover}(t)$, and 0 otherwise.

In this cover matrix, or *cover space*, an itemset XYZ represents a group of code table elements $X, Y, Z \in CT$ that co-occur in the covers of transactions in D. As such, *frequent* itemsets in $\mathscr{C}$ make for good code table candidates: as instead of writing multiple codes together many times, we can gain bits if we can write one short code for their combination instead.

Introducing (or removing) an element X to CT changes the cover matrix, however, both in numbers of columns, and values. As the co-occurrences change, so does what makes a good addition to the current CT. This suggests that to find good candidates, we have to iteratively search for frequent itemsets in cover space, updating for every change in CT.

Estimating candidate quality

As simple an observation it is, mining good itemsets in cover space to find good code table candidates is the key idea of this chapter. In the remainder, we will show that by iteratively considering such itemsets, we can optimise compression far beyond KRIMP, and do this much more quickly too. Before we can do so, we first have to discuss how to measure the quality of a candidate.

As we are optimising compression, the quality of a candidate X essentially is the gain in total compression $L(CT, D)$ we obtain when we add X to CT. The most straightforward approach is to calculate these gains for every candidate and then to select the best one. As such, every candidate X to be considered corresponds to the combination of code table elements identified by an itemset Y in $\mathscr{C}$. The items in Y are in fact indexes to elements in CT, i.e. $Y \subseteq \{1, \ldots, |CT|\}$. Hence, X is simply the union of the code table elements $X_i \in CT$ identified by the $i \in Y$, i.e. $X = \bigcup_{i \in Y} X_i$.

Next, once we know the gains in compression for every candidate, we simply locally optimise and accept that X into CT which maximises compression. Although powerful, this is expensive, since to calculate the gain for a candidate, we have to cover the database.

Instead, we can *estimate* the gain first, and only calculate exact gain for the best estimated candidate. To maximise compression, we observe we want candidates 1) with high *usage* (i.e. short codes), 2) replacing many codes in $\mathscr{C}$, while 3) not adding much complexity. The first two of these properties are simply approximated by finding frequent itemsets in $\mathscr{C}$.

However, we can say more on what itemsets we want: highly frequent sets of only few items. Besides that these sets will likely compress well, they add little complexity to CT. However, keeping in mind that by iteratively refining, if we consider too large itemsets in $\mathscr{C}$, we strongly reduce our search space and are more likely to end up in a local minimum. Moreover, calculating the gain in compression is most accurate for itemsets of length 2. Hence, we restrict the cardinality of the itemsets we find in $\mathscr{C}$ to length 2. Note however, that more specific code table elements, i.e. large itemsets in D, can (and are) be constructed in only few iterations.

Suppose X and Y are itemsets in CT. Let CT' be the code table after adding the union of these two itemsets, i.e. $CT' = CT \oplus (X \cup Y)$. We can estimate the *usage* of $X \cup Y$ in CT' as

$$|usage(X) \cap usage(Y)|\,,$$

where by $usage(X)$ we slightly abuse notation to refer to the *tid* list of transactions with X in their cover. Note this gives an upper bound to $usage(X \cup Y)$, where equality only holds if $X \cup Y$ is considered for covering right before when X or Y would be used. As code table elements are ordered first on length, and the union of two non-equal itemsets produces a longer set, this is implicitly enforced—making the bound quite tight.

Although useful, as a gain estimate it is quite rough: it disregards the effects on $L(CT \mid D)$ of adding $X \cup Y$ to CT, the increase of the code lengths of X and Y as their *usage* decreases, as well as the scaling effect on all other code lengths. We can improve our estimate by taking these effects into account as follows, directly estimating the total compressed size for adding a candidate.

Let ΔL be the difference in encoded size between CT and CT'; or, in other words, the gain in bits for candidate $X \cup Y$. It is easy to see that ΔL for the total encoded size

consists of the difference in compressed sizes of the database, and the difference between the encoded sizes of the code tables,

$$\begin{aligned}
&\Delta L(CT \oplus (X \cup Y), D) \\
&\quad = L(CT, D) - L(CT \oplus (X \cup Y), D) \\
&\quad = \Delta L(D \mid CT \oplus (X \cup Y)) + \Delta L(CT \oplus (X \cup Y) \mid D)\,.
\end{aligned}$$

For notational brevity, we use the lower case, x, of an itemset $X \in CT$ to indicate $usage(X)$, i.e. $x = usage(X)$. Then, let s be the sum of usages of CT, i.e. $s = \sum_{X \in CT} x$. Similarly, we use x' and s' for CT'. Using this notation, we can write the difference in bits between encoding D by either CT or CT' as follows,

$$\Delta L(D \mid CT \oplus (X \cup Y)) = \; s \log s - s' \log s' + xy' \log xy' - \sum_{\substack{C \in CT \\ c \neq c'}} (c \log c - c' \log c')\,,$$

and the difference in the model complexity as

$$\begin{aligned}
\Delta L(CT \oplus (X \cup Y) \mid D) = {} & \log xy' - L(X \cup Y \mid ST) \\
& + |CT| \log s - |CT'| \log s' + \sum_{\substack{C \in CT \\ c' \neq c \\ c'c \neq 0}} \log c' - \log c \\
& + \sum_{\substack{C \in CT \\ c' \neq c \\ c = 0}} \log c' - L(C \mid ST) + \sum_{\substack{C \in CT \\ c' \neq c \\ c' = 0}} L(C \mid ST) - \log c\,,
\end{aligned}$$

where $xy' = usage(X \cup Y)$ when using CT'. This means that we can express the gain in bits when adding $X \cup Y$ to CT only in terms of the usages of code table elements for which the usage changes between CT and CT'.

In practice, however, it is hard to predict how $usage(Z)$ for all $Z \in CT$ will develop when a new element $X \cup Y$ is inserted into CT. While we know the usages of $Z \in CT$ ordered above $X \cup Y$ remain static, a cascading effect may occur for those below it: in some $t \in D$, where some $Z \in CT$ with $Z \cap (X \cup Y) \neq \emptyset$ was previously part of the cover of t, $X \cup Y$ may now prevent Z from being used; hence changing *usage* of Z, as well as the *usage* of those Z' now used to cover $Z \setminus (X \cup Y)$ of t, which in turn can block other previously used elements, etc.—potentially changing all usages.

Although unpredictable, in practice the effect is not often dramatic. Hence, for practical reasons, for estimating the gain we can assume only the *usage* of X, Y, and $X \cup Y$ will change, using our above usage estimate for $X \cup Y$. For calculating the estimated gain in total compressed size, we then simply use $xy' = |usage(X) \cap usage(Y)|$, $x' = x - xy'$, $y\prime = y - xy'$ and $s' = s - xy'$. As such, we obtain a very easily calculable, and as we will see, generally accurate estimate of ΔL for $X \cup Y$.

Algorithm 4 The SLIM Algorithm

Input: A transaction database D over a set of items $\mathscr{I}$
Output: A heuristic solution to the Minimal Coding Set Problem, code table CT

1. $CT \leftarrow$ **Standard Code Table**(D)
2. **for** $F \in \{X \cup Y : X, Y \in CT\}$ in **Gain Order do**
3. $\quad CT_c \leftarrow (CT \oplus F)$ in **Standard Cover Order**
4. $\quad$ **if** $L(D, CT_c) < L(D, CT)$ **then**
5. $\quad\quad CT \leftarrow$ *post-prune*(CT_c)
6. $\quad$ **end if**
7. **end for**
8. **return** CT

5.4 Directly Mining Descriptive Patterns

We can now combine the above results, and construct the SLIM algorithm for mining heuristic solutions to the Minimal Coding Set Problem directly from data.[1]

We give the pseudo-code as Algorithm 4. SLIM starts with the singleton-only code table ST (line 1). Every iteration (2) we consider all pairwise combinations of $X, Y \in CT$ as candidates in **Gain Order**, i.e. descending on $\Delta L(CT \oplus (X \cup Y), D)$. Iteratively, we add a candidate to CT in **Standard Cover Order** (3), cover the data, and compute total encoded size (4). If compression improves, we accept the candidate, otherwise reject it. If accepted, we reconsider every element in CT to whether it still contributes towards compression (5), and update the candidate list (2). We continue considering pairwise combinations of $X, Y \in CT$ to refine the current code table until no candidate decreases the total compressed size, after which we are done.

Note that, if desired, extra constraints on individual candidates (e.g. a minimum support, or length) can be checked when constructing the candidate list, or before adding them to the code table at line 3.

In practice, we do not need to materialise all candidates on line 2. Instead, we traverse CT ordered on usage, employing branch-and-bound to find the $X \cup Y$ with highest estimated gain; as we traverse the elements descending on usage, we do not need to consider any element V or W with lower usage than the current best candidate $X \cup Y$. Moreover, suppose X is considered before Y. Therefore $usage(Y) \leq usage(X)$, and we can first bound using $usage(X \cup Y) = usage(X)$. Second, we can bound using $usage(X \cup Y) = usage(Y)$. Then, if this bound is met, we need to calculate the expected usage $usage(X \cup Y)$ by intersecting the usage lists of X and Y. Moreover, to speed up computation, we store the top-k best estimates, allowing us to quickly suggest the next-best candidate when a candidate is rejected.

SLIM is well-suited for any-time computation, as it iteratively refines the current code table. As such, it allows for interactive data analysis and time-budgeted computation, providing good intermediate results. Given a result, SLIM can simply continue refining.

Next, we analyse the complexity of SLIM. Considering the candidates (line 2) maximally takes $O(|CT|^2)$ steps. A code table for D could contain all $|\mathscr{F}|$ itemsets occurring in D. At worst, we re-evaluate each candidate $|\mathscr{F}|$ times. The complexity of

[1] SLIM is Dutch for *smart*, whereas KRIMP means *to shrink*.

steps 3–6 is $O(|\mathscr{F}| \times |D| \times |\mathscr{I}|)$. Together, the worst-case time-complexity is $O(|\mathscr{F}|^3 \times |D| \times |\mathscr{I}|)$. We will see in the experiments this estimate is quite pessimistic; in practice, code tables are small (ranging from 10s to 1000s) and SLIM evaluates up to 5 orders-of-magnitude fewer candidates than all $|\mathscr{F}|$ itemsets occurring in D.

Regarding memory complexity we can be brief. As we need to store a code table of maximally $|\mathscr{F}|$ itemsets, and the database D, memory complexity is $O(|\mathscr{F}| + |D|)$.

5.5 Related Work

Since the seminal paper by Agrawal and Srikant [2] on frequent pattern mining, a lot of research is aimed at reducing the pattern explosion; mining the most interesting and useful patterns in manageable amounts. The traditional approach is to mine concise representations for collections of patterns, either lossless, such as non-derivable [12] itemsets, or lossy, as for self-sufficient itemsets [91] and probabilistic summaries [93]. This is different from our approach in that we summarise data, instead of pattern collections.

Webb argues [90] not to condense set of mined patterns, but to rank patterns according to their statistical significance, and let the end-user consider the top-k. However, as patterns are considered individually, redundancy among significant patterns remains.

Considering patterns as binary features on rows, Knobbe and Ho [36], and Bringmann and Zimmermann [10], resp. exhaustively and heuristically select those groups of patterns by which data rows are partitioned optimally, using an external criterion such as joint-entropy or accuracy. Unlike SLIM, both post-process materialised collections of candidate patterns, and partition the data instead of summarising it.

Tilings [22] are sets of itemsets that together efficiently cover the data, and are hence strongly related to SETCOVER. Although tilings can be mined directly from data, as area is not (anti-)monotonic with set inclusion, efficiency is an issue. Related, Kontonasios and De Bie [37] propose a two-phase approach to select the most informative noisy tiles from a collection of fault-tolerant itemsets, using MDL and a maximum entropy data model. Both methods require a number of patterns to select, as well as a minimum area threshold.

SLIM is strongly related to the KRIMP algorithm [87]. Both aim at finding the set of itemsets that together describe the data best. KRIMP code tables have been shown to capture data distributions very well, and have been used successfully for a wide range of data mining tasks [41, 86, 87]. KRIMP post-processes a candidate collection, filtering it in a static order. By iteratively considering the current data description, SLIM avoids redundant patterns, explores a larger search space, does not mine and sort huge numbers of candidates, and does not require a minimal support threshold.

Recently, Siebes and Kersten [69] proposed the GROEI algorithm for finding the best k-element KRIMP code table by beam search. By considering a much larger search space, they improve over KRIMP at expense of efficiency—for beam-width 1 it coincides with KRAMP—and hence only small datasets are considered. Although beyond the scope of this thesis, the SLIM search strategy and estimation heuristics can likely speed up GROEI significantly.

The PACK algorithm [76] makes a connection between decision trees and itemsets, mines good decision trees, and returns the itemsets that follow from these. It can mine its trees either from a candidate itemset collection, or directly from the data. Unlike here, PACK considers the data 0/1 symmetrically. As a result, it requires fewer bits, but returns many more itemsets.

Wang and Partharsarthy [89] incrementally build probabilistic models for predicting itemset frequencies. Iteratively they update the model by those itemsets for which the estimate deviated more than the threshold. For efficiency, itemsets are considered in level-wise batches. Mampaey et al. [48] propose a convex heuristic to efficiently iteratively find the least-well predicted itemset overall, using BIC to control complexity. As constructing and querying maximum entropy models is computationally very expensive, only high-level summaries can be mined. Furthermore, by only considering frequency, co-occuring patterns cannot be detected [37].

5.6 Experiments

Here we experimentally evaluate SLIM, its heuristics, and the quality of the discovered code tables.

Setup

We implemented a prototype in C++, and provide the source code for research purposes.[2]

We use the shorthand notation $L\%$ to denote the relative compressed size of D,

$$\frac{L(D, CT)}{L(D, ST)}\%,$$

wherever D is clear from context. Lower values indicate better compression and are preferred from a MDL point of view.

As candidates $\mathscr{F}$ for KRIMP, we use all frequent itemsets mined at the *minsup* thresholds depicted in Table 5.2. These thresholds were chosen as low as feasible, while making the processing finish within 24 hours. KRIMP includes AFOPT one of the fastest miners from the FIMI repository [87]. Timings reported for KRIMP include mining and sorting of the candidate collections.

All experiments were conducted as single-threaded runs on Linux machines with Intel Xeon X5650 processors (2.66GHz) and 12 GB of memory.

Datasets

We consider a wide range of benchmark and real datasets. The base statistics for each are depicted in Table 5.1. We show for each database the number of attributes, number of transactions, the density (relative number of 1s), and the number of classes.

From the LUCS/KDD dataset repository we take some of the largest and most dense databases. From the FIMI repository we use the *Accidents*, *BMS* and *Pumsb*

[2] `http://adrem.ua.ac.be/implementations/`

Table 5.1: General statistics.

Dataset	$\|D\|$	$\|\mathscr{I}\|$	%1s	$\|\mathscr{K}\|$
Abstracts	859	3933	1.2	-
Accidents	340183	468	7.2	-
Adult	48842	97	15.3	2
BMS-pos	515597	1657	0.4	-
BMS-webview 1	59602	497	0.5	-
Chess (k-k)	3196	75	49.3	2
Chess (kr-k)	28056	58	12.1	18
Connect-4	67557	129	33.3	3
DNA amplification	4590	391	1.5	-
Ionosphere	351	157	22.3	2
Letter recognition	20000	102	16.7	26
Mammals	2183	121	20.5	-
MCADD	31924	198	11.1	2
Mushroom	8124	119	19.3	2
Pen digits	10992	86	19.8	10
Plants	34781	70	12.4	-
Pumsb	49046	2113	3.5	-
Pumsbstar	49046	2088	2.4	-
Waveform	5000	101	21.8	3

Shown are the number of transactions, attributes, density (in percentage of 1s) and number of classes (if any).

datasets. The *Chess (kr–k)* and *Plants* datasets were obtained from the UCI repository.

We use the real *Mammals* presence and *DNA Amplification* databases. The former consists of presence records of European mammals[3] within geographical areas of 50×50 kilometers [53]. The latter contains DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of advanced tumours [55].

From the Antwerp University Hospital we obtained *MCADD* data [73]. Medium-Chain Acyl-coenzyme A Dehydrogenase Deficiency (MCADD) is a deficiency all newborns are screened for [81].

The *Abstracts* dataset contains the abstracts of all accepted papers at the ICDM conference up to 2007, where words are stemmed and stop words removed [37].

In this chapter we mainly focus on the above data sets, however, we will reuse the (smaller) data sets from previous chapter, depicted in Table 4.1, to analyse two

[3]The full version of the mammal dataset is available for research purposes upon request from the Societas Europaea Mammalogica. `http://www.european-mammals.org`

Table 5.2: Overview statistics.

	KRIMP				SLIM		
Dataset	*minsup*	$L\%$	$\|CT\|$	$\|\mathscr{F}\|$	$L\%$	$\|CT\|$	$\|\mathscr{F}\|$
Abstracts	3	91.9	1102	6 M	**84.3**	1815	13 M
Accidents	50000	55.1	1583	3 M	**31.1**	2018	21 k
Adult	1	24.4	1303	58 M	**22.8**	1201	199 k
BMS-pos	100	**81.7**	9478	6 M	83.1	1964	15 k
BMS-webview 1	35	85.9	718	1 M	**84.0**	965	2 M
Chess (k-k)	500	30.0	275	846 M	**14.7**	292	20 k
Chess (kr-k)	1	61.6	1684	373 k	**57.5**	1060	28 k
Connect-4	40000	42.9	56	24 M	**12.3**	1670	297 k
DNA amplification	9	36.7	326	312 M	**35.7**	359	67 k
Ionosphere	35	59.8	170	226 M	**49.7**	240	294 k
Letter recognition	1	35.7	1780	581 M	**33.4**	1599	521 k
Mammals	200	48.1	316	94 M	**39.9**	434	235 k
MCADD	35	55.4	2280	2 M	**51.0**	4067	924 k
Mushroom	1	20.5	442	6 G	**18.5**	340	16 k
Pen digits	1	42.2	1247	459 M	**39.4**	1347	394 k
Plants	2000	46.4	511	913 k	**36.1**	840	179 k
Pumsb	35000	70.0	175	2 M	**19.1**	3299	151 k
Pumsbstar	12500	56.0	331	2 M	**25.1**	4274	383 k
Waveform	5	44.5	921	466 M	**39.0**	734	134 k

For KRIMP we give relative compression ($L\%$), number of non-singleton code table elements, and number of candidates for the given *minsup* threshold. For SLIM we give relative compression ($L\%$), number of non-singleton code table elements, and number of evaluated candidates.

more computationally intensive greedy optimisation strategies and to compare the experiments regarding anomaly detection.

Compression

First we investigate how well SLIM describes data. In Table 5.2 we give the relative compression rates ($L\%$) for both SLIM and KRIMP. To ease interpretation we also plot these in Figure 5.1. The shorter the bars in the plot, the shorter the discovered descriptions of the data.

Overall, we see SLIM outperforms KRIMP with an average gain in compression ratio of 11%, up to over 50% for *Pumsb*. The only exception is *BMS-pos*, for which both methods fail to find succinct descriptions.

Analysing these results in more detail, we see that, unsurprisingly, the largest improvements are made on the large and/or dense datasets, such as *Accidents*,

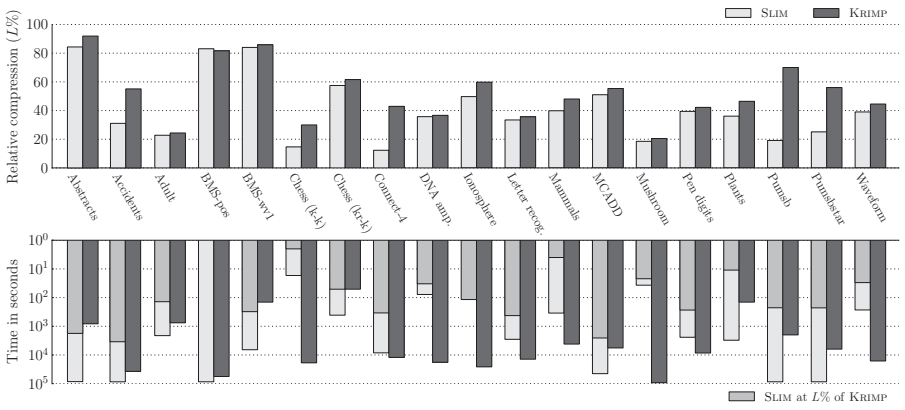


Figure 5.1: Comparing SLIM and KRIMP: relative compression (L%) (top), overall running time in seconds (bottom), and time needed by SLIM to reach the compression attained by KRIMP (mid-grey). Note that besides obtaining better compression, SLIM is nearly always faster than the optimised KRIMP implementation.

Connect-4, and *Ionosphere*. By their characteristics, these datasets give rise to very large numbers of patterns, and hence can only be considered by KRIMP if we set *minsup* relatively high—implicitly limiting the detail at which KRIMP can describe the data.

The ability to compress a dataset depends on the amount of recognisable structure. For sparse datasets, like *Abstracts*, and the two *BMS* datasets, neither SLIM nor KRIMP can identify much structure, whereas SLIM can describe *Pumsb(star)* quite succinctly by considering lower-frequency itemsets. For dense data, such as *Chess (k-k)*, *Connect-4*, and *Mushroom*, both algorithms ably find structure.

When we look at the number of (non-singleton) itemsets in CT, we see very similar results. In general, for both SLIM and KRIMP, depending on the data, the code tables contain between a hundred up to a few thousand itemsets. Three datasets, *Connect-4* and *Pumsb-(star)*, stand-out, with SLIM returning 10 times more patterns. However, for these KRIMP can only run with very high *minsup*—whereas SLIM does not have this restriction, and can better capture the structure by using more fine-grained patterns.

Greedy vs. Greedy

Next, we compare SLIM to two variants of the powerful standard greedy optimisation algorithm for complex combinatorial problems. KRAMP is a variant of KRIMP that in each iteration chooses that F out of all $\mathscr{F}$ that locally maximises compression. Analogously, SLAM is a variant of SLIM calculating exact compression gain to order the candidates at every iteration—instead of estimating it heuristically.

Clearly, KRAMP and SLAM are computationally expensive. Hence, we first compare on some small, well-known UCI benchmark datasets: *Anneal, Breast, Heart, Iris, Led7, Nursery, Page blocks, Pima, Tic-tac-toe* and *Wine*. All these datasets are easily mined and processed using a *minsup* threshold of 1. Comparing the total

Table 5.3: Greedy vs. Greedy

Dataset	KRIMP	SLIM	SLAM	KRAMP
Anneal	34.6	31.3	31.8	32.4
Breast	16.9	15.6	15.6	16.6
Heart	55.8	48.1	48.0	50.5
Iris	45.7	45.5	45.5	45.6
Led7	28.5	27.4	27.4	28.3
Nursery	45.5	43.2	43.1	44.1
Page blocks	5.0	5.0	4.9	5.0
Pima	34.1	30.9	30.3	31.9
Tic-tac-toe	62.4	49.5	47.0	52.1
Wine	72.8	71.0	70.2	71.5
average	40.1	36.8	36.4	37.8
Adult	24.4	22.8	22.5	22.9
Chess (k-k)	30.0	14.7	14.9	
Chess (kr-k)	61.6	57.5	57.5	
DNA amplification	36.7	35.7	35.7	
Ionosphere	59.8	49.7	48.9	
Letter recognition	35.7	33.4	32.6	
Mushroom	20.5	18.5	18.8	
Pen digits	42.2	39.4	38.4	
Waveform	44.5	39.0	38.5	
average	39.5	34.5	34.2	
Connect-4	42.9	12.3	22.1	
Accidents	55.1	31.1	43.3	

Comparing relative compression ($L\%$) for the different greedy algorithms. First block shows the results for the smallest datasets, i.e. where both SLAM and KRAMP finish in less than 24 hours. Second block shows the result given SLIM one day and SLAM one week to converge. In the last two blocks, we give some results for which SLAM is not converged within one week or where both SLIM and SLAM still need computation time after the previous mentioned time constraints. Only for *Adult* we let KRAMP run for two months.

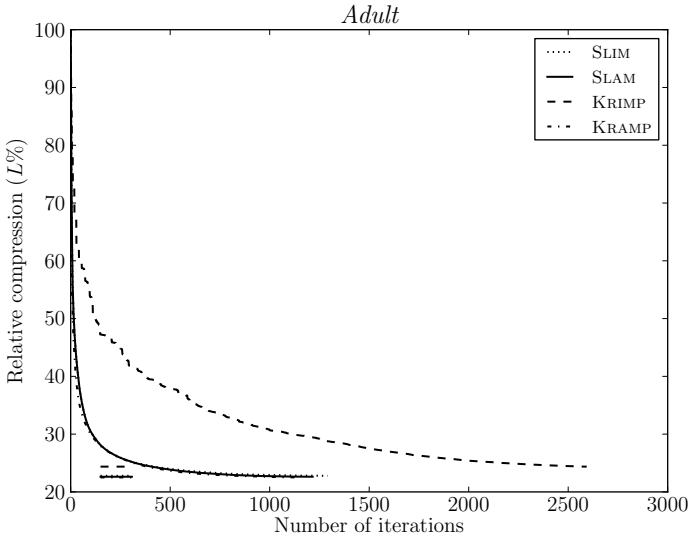


Figure 5.2: Convergence plot for *Adult*. The lines in lower left corner show the final attained relative compression.

compressed sizes in Table 5.3, we observe that SLAM is always ranked best, SLIM second, KRAMP third and KRIMP last, with an average relative compression (L%) of respectively 36.4, 36.8, 37.8 and 40.1. Note that although KRAMP considers the largest search-space, it does not always obtain the best result. In the remainder, we do not consider these datasets further.

When we consider some larger databases, we see the same pattern: SLAM obtains the best average L% of 34.2, SLIM a close second at 34.5, and KRIMP is behind with 39.5—KRAMP does not finish within reasonable time.

Last, using *Adult* as a typical example, we plot the development of L% per itemset accepted into CT as Figure 5.2. As the plot shows, SLIM closely follows SLAM and KRAMP, quickly converging to good compression—much more directly than KRIMP. Note that as itemsets can be pruned from CT, the final x-coordinate does not necessarily match $|CT|$ in Table 5.2. Further, we note that while SLIM only needs 35 minutes to converge, SLAM requires one week, and KRAMP two months. We will discuss convergence and runtime in more detail below.

Number of Candidates

Next, we compare the number of instantiated candidates $|\mathscr{F}|$, shown in Table 5.2. This is the number of itemsets for which we calculate the total compressed size by covering the data. For KRIMP this equals the number of frequent itemsets at the listed *minsup* threshold. For SLIM this reflects the number of materialised unions of code table elements. As Table 5.2 shows, SLIM evaluates 2 orders-of-magnitude fewer candidates. In general, SLIM considers between 10 thousand and 10 million itemsets, whereas KRIMP processes millions to billions of itemsets.

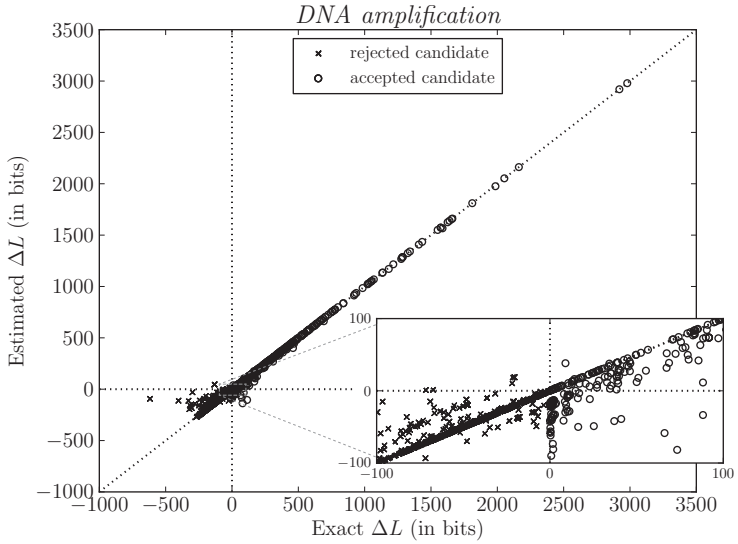


Figure 5.3: Correlation plot of the estimated versus exact ΔL (in bits) on the *DNA amplification* dataset.

Only for *Abstracts* and *BMS-webview 1* SLIM evaluates more candidates than KRIMP—datasets for which a minute decrease in *minsup* leads to an explosion of candidates. As such, also for this sparse data SLIM can consider candidates at lower support than KRIMP can handle, while for the other datasets SLIM only requires a fraction to reach better compression.

Timings and convergence

We now inspect run-times and convergence. We plot the total wall-clock running time for SLIM and KRIMP as the bottom bar plot of Figure 5.1. For KRIMP (darkest bars), this includes mining frequent itemsets, sorting, and selecting from them. For SLIM we mark two timestamps: in mid-grey we show the time it takes SLIM to overtake KRIMP; the light bars display the time to convergence, with a maximum run-time of 24 hours.

First we inspect how long SLIM requires to match the compression of KRIMP. We see that for 16 datasets SLIM reaches this point faster than KRIMP—in fact, several orders-of-magnitude faster for many of these, particularly for dense datasets. For *BMS-pos*, the bar is not shown, as SLIM does not reach the same compression. For *Ionosphere* the bar is simply not visible, as SLIM overtakes KRIMP in less than one second. As such, SLIM is generally much faster than KRIMP in obtaining KRIMP-level descriptions.

Second, we compare overall runtime. We see SLIM is still faster than KRIMP for 9 datasets, including huge improvements for *Chess (k-k)*, *DNA amplification*, *Ionosphere* and *Mushroom*. For the other datasets, the current implementation requires more time, yet it acquires much more succinct descriptions.

To inspect these cases, we again consider Figure 5.2. By optimising (estimated) gain the compressed size converges much faster for SLIM than for KRIMP. After the early large gain, the line quickly levels. Although not converged, now only very few bits are gained per candidate. This goes general, where for *Pumbsb(star)* runtime is further increased by the relatively large code table. As SLIM is worst-case quadratic in $|CT|$, the tail of convergence is where most time is invested: CT grows, while few candidates can be pruned. Caching evaluations between iterations will likely speed up the implementation. A more efficient encoding would make selection more strict, providing better descriptions *and* do away with the small-gain candidates.

In Figure 5.3 we plot the correlation between our estimate of ΔL to the actual gain in total compressed size. We see the two show strong correlation, as most of the measurements lie on the diagonal, especially for gain (upper right quadrant). Moreover, we see almost no false positives (upper left quadrant), and only few examples where few bits are gained while we estimated small loss (lower right quadrant). This strong correlation explains why the convergence of SLIM and SLAM are similar in Figure 5.2.

Classification

Above, we saw SLIM describes data more succinct than KRIMP. We here independently validate how well it captures data distributions by classification: KRIMP showed performance on par with 6 state-of-the-art classifiers on a wide range of datasets [87]. If SLIM performs at least as well, we can say it mines code tables at least as characteristic for the data.

For this, we reuse the simple classification scheme based on code tables [87]. To use it, we need a code table per class. To build those we split the database according to class, after which the class labels are removed from all transactions. We then apply SLIM and KRIMP to each of these class-databases, resulting in a code table per class. When the compressors have been constructed, classifying a transaction t is trivial: simply assign the class label belonging to the code table providing the minimal encoded length for t.

We measure performance by accuracy, the percentage of true positives on the test data. All reported results have been obtained using 10-fold cross-validation. Note that we do not maximise accuracy by choosing good pairings between intermediately reported code tables [87], but simply use the final code tables.

We give the accuracy scores as Figure 5.4. We observe high similarity between SLIM and KRIMP, and that on average SLIM performs slightly better. A pair-wise Student t-test reveals that only the results on the *Connect-4* and *Chess (kr-k)* differ significantly, at significance levels lower than 0.1%, respectively in favour of SLIM and KRIMP. The performance for *Chess (kr-k)* is due to SLIM much better describing the large classes, whereas the more general KRIMP code tables balance better against those for the small classes.

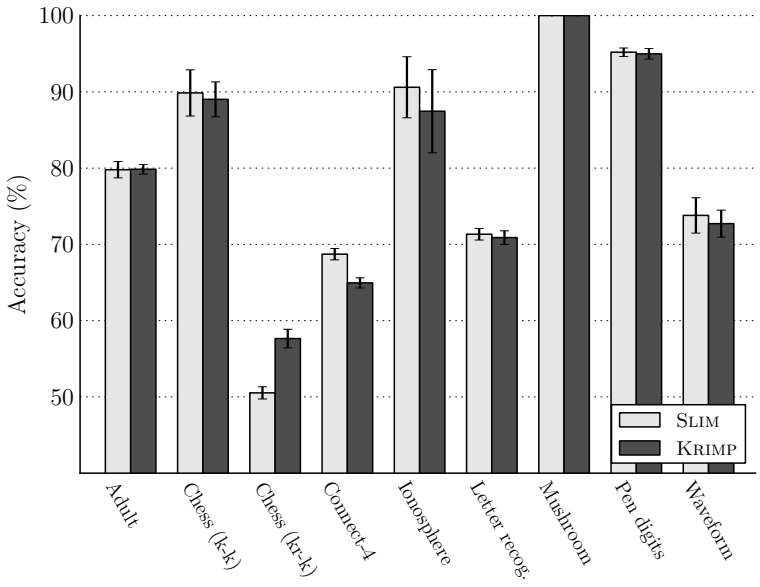


Figure 5.4: Comparing accuracy on classification tasks.

Anomaly Detection

Chapter 4 discusses how compression can be used to identify cases that differ from the norm. Informally said, by building a compressor on a database, we can decide whether a sample is an anomaly by its encoded length: if more bits are required than expected, the sample is likely anomalous. Experiments show KRIMP performs on par with the best anomaly detection methods for binary and categorical data.

To see whether SLIM is as useful as KRIMP for detecting anomalies, we consider both the results on benchmark datasets and a case study on detecting carriers of MCADD, a rare metabolic disease [81].

To compare SLIM and KRIMP, we use the AUC, i.e. area under the ROC curve, as it is independent of actual decision thresholds. Table 5.4 shows the AUC scores averaged over 10-folds and each of the classes of each dataset. We see, and it is confirmed by pairwise Student's t-tests at α-level 5%, that, in general, their is no difference in performance between SLIM and that of KRIMP. However, SLIM consistently outperforms KRIMP on the datasets for which the latter is mined using a *minsup* higher that 1: *Chess (k-k)*, *Connect-4*, *Ionosphere*, *MCADD* and *Waveform*. This confirms the claim made in Section 4.5 that some characteristic patterns might be missing from KRIMP code tables if memory of time constraints do not allow us to mine for frequent item sets at low *minsup*. In particular, SLIM improves, in less than 10 seconds, the results on the *Chess (k-k)* data set significantly when comparing to the scores of the other classifiers in Table 4.2.

Table 5.4: One-Class Classification

Dataset	SLIM	KRIMP	p-value
Adult	67.02 ± 2.81	68.72 ± 2.64	0.18
Anneal	**97.62 ± 2.08**	95.32 ± 2.17	0.03
Breast	84.93 ± 12.06	86.93 ± 8.73	0.68
Chess (k-k)	**88.41 ± 2.95**	68.53 ± 0.46	0.00
Chess (kr-k)	94.57 ± 5.92	94.84 ± 5.05	0.91
Connect-4	73.72 ± 4.51	70.32 ± 5.28	0.14
Heart	59.13 ± 10.24	65.65 ± 8.01	0.13
Ionosphere	**69.48 ± 6.82**	63.65 ± 4.67	0.04
Iris	96.60 ± 1.23	94.73 ± 3.61	0.14
Led7	91.35 ± 3.58	91.50 ± 3.46	0.93
Letter recognition	93.82 ± 1.83	92.22 ± 1.73	0.06
MCADD	95.11 ± 4.47	92.08 ± 7.74	0.30
Mushroom	100.00 ± 0.00	100.00 ± 0.00	1.00
Nursery	96.05 ± 3.74	87.62 ± 20.49	0.22
Page blocks	50.67 ± 21.43	51.02 ± 21.54	0.97
Pen digits	99.10 ± 0.61	98.58 ± 0.74	0.10
Pima	51.18 ± 20.62	50.69 ± 20.87	0.96
Tic-tac-toe	88.92 ± 5.43	88.08 ± 6.61	0.76
Typist	86.88 ± 0.91	86.75 ± 1.82	0.84
Waveform	82.71 ± 9.87	80.14 ± 10.44	0.17
Wine	95.17 ± 2.97	94.23 ± 1.58	0.39
average	83.93 ± 15.73	81.98 ± 15.34	0.67

AUC scores for the benchmark classification datasets in Table 4.1 and 5.1 for the anomaly detection tasks using OC^3. Shown are, per dataset, mean and standard deviation of the average AUC score over the classes. The KRIMP-compressor in OC^3 ran using all frequent itemsets above the *minsup* values in Table 5.2 as candidates. The last column indicates the p-values using a pairwise Student's t-test and the AUC scores that differ significantly at α-level 5% are highlighted in bold.

Moreover, repeated analysis of the case-study on the *MCADD* dataset show SLIM provides similar performance to KRIMP: all 8 anomalous cases are ranked among the top-15 in both code tables, and the obtained performance indicators (100% sensitivity, 99.9% specificity and a predictive value of 53.3%) correspond with the state-of-the-art results [81].

The highly-ranked non-MCADD cases show combinations of attribute-values very different from the general population, and are therefore abnormal by definition. Analysing the encoding of normal cases reveals that the code tables correctly identify attribute-value combinations commonly used in diagnostics by experts [81].

Manual Inspection

Finally, we subjectively evaluate the selected itemsets. To this end, we take ICDM *Abstracts* dataset. Considering the top-k itemsets with highest *usage* we see SLIM and KRIMP provide highly similar results: both provide patterns related to topics in data mining—e.g. 'mine association rules [in] databases', 'support vector machines (SVM)', 'algorithm [to] mine frequent patterns'.

One can argue, however, that experts should not only look at the top-ranked itemsets, as the patterns are selected together to describe the data. When we browse the code tables a whole, we see SLIM and KRIMP select roughly the same patterns of 2 to 5 items. As KRIMP only considers patterns of at least *minsup* occurrences, in this data it does not consider more specific itemsets, whereas SLIM may consider any itemset in the data.

For this dataset, SLIM selects a few highly specific patterns, such as *'femal, ecologist, jane, goodal, chimpanze, pan, troglodyt, gomb, park, tanzania, riplei, stuctur'*. Inspection shows these itemsets all represent small groups of papers sharing (domain) specific terminology—an application paper in this case—that is not used in any of the other abstracts. As such, these itemsets make sense from both interpretation, and MDL perspective; since the likelihood of these words is very low, yet they strongly interact, and hence can best be described using a single (rare) pattern, saving bits by not requiring codes for the individual rare words.

Note however, that if such level of detail is not desired, a domain expert can prune the search space by specifying additional constraints, e.g. by specifying *minsup*, on the candidates SLIM may consider.

5.7 Discussion

The experiments show SLIM finds sets of itemsets that describe the data well, characterising it in detail. In particular on large and dense datasets, SLIM code tables obtain tens of percents better compression ratio than KRIMP. Classification results show high accuracy, verifying that high-quality pattern sets are discovered.

Dynamically reconsidering the set of candidates while traversing the space leads to better compression. In particular, SLIM closely approximates the expensive greedy algorithms KRAMP and SLAM, that select the best candidate of all current candidates. By employing a branch-and-bound strategy using an efficient and tight heuristic, SLIM is much more efficient. The good convergence adds to its any-time property, providing users good results within a time-budget, while allowing further refinements given more time.

SLIM evaluates up to 5 orders-of-magnitude fewer candidates than KRIMP, while obtaining more succinct descriptions. Moreover, although the KRIMP implementation is optimised, the prototype implementation of SLIM is often quicker too. SLIM can be optimised in many ways. For example, we currently do not re-use information from previous iterations; nor do we cache whether items co-occur at all. Furthermore, SLIM is trivial to parallelise, including parallel branch-and-bound search, evaluation of top-ranked candidates, and covering the data.

Although the exact effects on *usage* of a change in CT can only be calculated by covering the data, better estimates of ΔL may lead to a further decrease in the number of evaluated candidates. Moreover, if the estimate would be sub modular, efficient optimisation strategies with provable bounds could be employed [57].

Our main goal is to provide a faster alternative to KRIMP that can operate on large and dense data, while finding even better sets of patterns. Both for describing succinctly, and classification we have seen SLIM is indeed at least as good as KRIMP. Further research needs to verify whether SLIM can also match or surpass KRIMP on other data mining tasks [41, 73, 86].

SLIM, SLAM and KRAMP all spend much of their time searching for candidates that only lead to minute improvements in compression. A natural heuristic to stop search early, without much expected loss of quality, would be to stop as soon as the gain estimate becomes negative or when a finite difference approximation of the derivative of $L\%$ indicates we reach a plateau. Another option is to make selection more strict by refining the encoding model.

Although beyond the scope of this dissertation, the encoding model can be improved in several ways. First, by allowing overlap during covering we can likely gain compression, although covering becomes more complex. Also, dynamic codes could be used; taking into account what itemsets can or cannot be used to encode the remainder of t. These changes will make selection more strict, increasing convergence, possibly avoiding small-gain candidates, yet will make encoding more complex. Future work includes investigating whether this indeed leads to better, i.e. more useful, code tables.

5.8 Conclusion

In this chapter, we introduced SLIM, an any-time algorithm for mining small, useful, high-quality sets of patterns directly from data. We use MDL to identify the best set of itemsets as that set that describes the data best. To approximate this optimum, we iteratively consider what refinement provides most gain—estimating quality using a light-weight and accurate heuristic. Importantly, SLIM is completely parameter-free.

Experiments show that SLIM is able to discover high-quality pattern sets, resulting in high compression rates and high accuracy scores. Furthermore, SLIM closely approximates the common greedy approach of selecting the best candidate overall, while being several orders of magnitude faster.

CHAPTER 6

Conclusions

In this thesis we explored several topics related to the detection and characterisation of anomalies in data. The research in Chapter 2 and 3 was motivated by specific real world problems, i.e. the identification of anomalies when monitoring the production processes in a chemical plant and the detection of vandalism on Wikipedia. In Chapter 4 and 5 we presented a more general technique to detect anomalies in binary or transaction data using an on itemsets based compressor and we applied it on a case study to screen newborns for a rare disease called MCADD.

In addition to the inherent changes to cope with the application and data-dependent specifications, we shifted from a machine learning to a more data mining perspective to identify and characterise anomalies in data. We made a transition from a more predictive inspired principle, Structural Risk Minimization (SRM), towards a more descriptive oriented principle, Minimal Description Length (MDL). Both model selection principles are mathematically sound and tend to trade-off complexity of the model and the quality of fitting/describing the training data.

In Chapter 2 we use a specific instantiation of SRM, namely Support Vector Data Description (SVDD), that balances the size of the hypersphere to the proportion of training examples that fall outside this sphere. Since the description of this hypersphere only depends on the support vectors, the training examples lying on the circumference, the model can be seen as a (lossy) compression of the data. In MDL, however, compression of the data is primary, as one minimises both the number of bits needed to describe the models, itemset-based code tables in Chapter 4 and 5, and the number of bits needed to encode the training data using this model.

Being predictive in nature, it is not surprising that SVDD performs better than OC^3 in Chapter 4. Moreover, Vapnik proved that the asymptotic bounds on the prediction risk are less tight for MDL than SRM, albeit that the prediction risk decreases as the compression ratio increases [83]. The transition to MDL, however, and more specifically to pattern-based compressors, is motivated by the need to detect anomalies using a parameter-free approach and to provide the data analyst with insight for the anomalousness.

Despite the good performance and the interesting property that the unbound support vectors indicate possible anomalies in the training data, SVDD constructs a global model that is hard to interpret. The classification of a new example tells us something about the (dis)similarity to some of the samples, objects or transactions in the training data, but does not give justification of the decision. Working with kernels in possibly infinite Hilbert spaces makes it hard to transfer knowledge back to the input space. Only in case of simple kernels, like a linear or polynomial kernel, it is straightforward, but tedious, to write and interpret the decision in terms of the input features. Moreover, transforming the input data prior to applying the kernel trick, as in Chapter 2, complicates analysis even more.

To allow easier analysis of the decisions and inspired by the good results on detecting spam in e-mails and on weblogs, we opted for using Naive Bayes and Prediction by Partial Matching respectively to detect vandalism on Wikipedia in Chapter 3. Although it should be possible to inspect the weighted n-grams, the PPM models produced using an off-the-shelf statistical compressor for sequences did not allow for such inspection.

In Chapter 4, relying on the state of the art itemset-based compressor KRIMP, we put a step forward to detect anomalies in binary or transaction data using an interpretable global model with good generalisation capabilities. Moreover, the local compressing blocks, the itemsets, remain visible throughout the compression of the data and are gathered in a compact code table. This allows a human expert to analyse decisions, either manually or guided with the technique presented in Chapter 4.

The code tables used Chapter 4 are, however, attained after post-processing a vast amount of possible patterns. To obtain these descriptive local patterns without requiring a pre-mined candidate collection, and thus to make the parameter-free analysis of dense and large datasets more feasible, we developed in Chapter 5 an any-time algorithm SLIM.

6.1 Contributions

The main contributions of the research presented in this thesis can be summarised as follows.

- In Chapter 2 we presented an algorithm for detecting anomalies in spatio/temporal data using one-class support vector machines. The tensor product combination of a time (or spatial) kernel and a data kernel, enables us to exploit vicinity relations in both time (or space) and data. A more general voting scheme, which combines an ensemble of SVDDs, enables us to incorporate historical information provided by different time-delay embeddings without

suffering loss of robustness or over-fitting. In case of temporal data, the algorithm detects anomalies due to time-shifts, abnormal peaks and phases that take too long.

- In Chapter 3 we demonstrated that the detection rate of the current handcrafted rule-based systems to fight vandalism on Wikipedia can already be improved employing a machine learning approach relying on a straightforward feature representation and a set of noisy labeled training examples.
- In Chapter 4 we provided a novel approach to anomaly detection for binary or transaction data. By relying on pattern-based compression, our method allows for detailed inspection and characterisation of decisions, both by showing which patterns were recognised in the example, as well as by checking whether small changes affect the decision. Given a few prototype outliers, our method can reliably estimate the decision landscape. This allows the user to verify whether the outliers can be detected at all or to make an informed choice for the decision threshold.
- In Chapter 5 we introduced an efficient, one-phase, any-time algorithm for mining high-quality data descriptions directly from transaction data. Pattern sets are constructed in a bottom-up fashion, iteratively joining co-occurring patterns such that compression is maximised, resulting in a closer approximation of the optimal set of patterns. Moreover, interweaving candidate generation and search space traversal allows us to reduce the number of evaluated candidates by several orders of magnitude and to provide more detail when necessary. To further improve efficiency, we employed a simple yet accurate heuristic to estimate the gain or cost of introducing a candidate. In particular on large and dense datasets SLIM provides a faster alternative to KRIMP that is able to describe the data better, characterising it in more detail.

6.2 Outlook

The discussion sections in each of the previous chapters outline opportunities for future research. In this section, we provide a general view on more fundamental problems that could be tackled in future work.

Despite the fact that we only have to model the norm, we could argue that anomaly detection is a more difficult problem than the two thoroughly studied predictive modeling tasks, classification and regression, and therefore deserves closer attention. First, we mostly have information about the normal scenarios and possess limited information about the anomalies. Second, given the limited information of the anomalies, common methods to predict the generalisation risk or model selection techniques cannot be applied directly. Generating artificial anomalies based on a prototype is one example of transforming the problem into a more traditional setting and to be able to apply standard methods. This is, however, not without risk, and more intensive research is needed to get insights in these risks and to provide answers to more general questions. Under which assumptions or conditions will a particular anomaly detection method work? And, more importantly, when will it fail? What kind of anomalies are we likely to detect and for which do we remain blind?

Moreover, in this thesis we only considered static normal models to detect and explain anomalies, and completely ignored the dynamic environments we are typically working in. In practice, an anomaly detection technique, and any other data mining task, cannot be treated in isolation. When new samples become available, the models should be updated accordingly to take into account the new information. Although building and verifying models to detect outliers seems initially harder, the same difficulties, typical for modeling in general, arise when put to practice. When things go wrong, for example when we fail to detect misbehaviour in a production process or if we produce too many false alarms, models should be updated accordingly.

Reacting upon and discovering new knowledge is easier if understandable models and explanations for unsuspected patterns are available. Providing this in dialog with the end-user remains one of the key challenges in the heterogeneous field of (interactive) data mining.

Bibliography

[1] B. T. Adler and L. de Alfaro. A content-driven reputation system for the Wikipedia. In *Proceedings of the 16th International Conference on World Wide Web (WWW), Banff, Alberta, Canada*, pages 261–270. ACM, 2007.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[3] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley & Sons, 3rd edition, 1994.

[4] R. Bathoorn, A. Koopman, and A. Siebes. Reducing the frequent pattern set. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops (ICDMW)*, pages 55–59, Washington, DC, 2006.

[5] C. Baumgartner, C. Böhm, and D. Baumgartner. Modelling of classification rules on metabolic patterns including machine learning and expert knowledge. *Journal of Biomedical Informatics*, 38(2):89–98, 2005.

[6] R. J. Bolton and D. J. Hand. Statistical fraud detection: A review. *Statistical Science*, 17(3):235–249, 2002.

[7] A. Bratko. PSMSLib: probabilistic sequence modeling shared library. Software available at `http://ai.ijs.si/andrej/psmslib.html`, 2006.

[8] A. Bratko, G. V. Cormack, B. Filipič, T. R. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, 7(Dec):2673–2698, 2006.

[9] S. Brin. Extracting patterns and relations from the world wide web. In *Proceedings of the International Workshop on the World Wide Web and Databases*, pages 172–183, 1998.

[10] B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM), Omaha, NE*, pages 63–72, 2007.

[11] L. S. Buriol, C. Castillo, D. Donato, S. Leonardi, and M. Stefano. Temporal analysis of the wikigraph. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), Hong Kong, China*, pages 45–51, 2006.

[12] T. Calders and B. Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

[13] J. Carter. ClueBot and vandalism on Wikipedia. Available at `http://www.acm.uiuc.edu/~carter11/ClueBot.pdf`, 2007.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15, 2009.

[15] F. Coenen. The LUCS-KDD discretised/normalised ARM and CARM data library, 2003. Software available at `http://www.csc.liv.ac.uk/~frans/KDD/Software/`.

[16] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

[17] M. Davy, F. Desobry, A. Gretton, and C. Doncarli. An online support vector machine for abnormal events detection. *Signal Processing*, 1(1):1–17, 2005.

[18] N. Dom, D. Knapen, D. Benoot, I. Nobels, and R. Blust. Aquatic multi-species acute toxicity of (chlorinated) anilines: Experimental versus predicted data. *Chemosphere*, 81(2):177–186, 2010.

[19] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India*, pages 1606–1611, 2007.

[20] S. García-Muñoz, T. Kourti, and J. F. MacGregor. Model predictive monitoring for batch processes. *Industrial Engineering Chemistry Research*, 43(18):5929–5941, 2004.

[21] A. B. Gardner, A. M. Krieger, G. Vachtsevanos, and B. Litt. One-class novelty detection for seizure analysis from intracranial EEG. *Journal of Machine Learning Research*, 7(Jun):1025–1044, 2006.

[22] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Proceedings of the 7th International Conference on Discovery Science (DS), Padova, Italy*, pages 278–289, 2004.

[23] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

[24] P. D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[25] D. J. Hand, P. Smyth, and H. Mannila. *Principles of data mining*. MIT Press, 2001.

[26] M. Hart, R. Johnson, and A. Stent. More content - less control: Access control in the web 2.0. In *Proceedings of the IEEE Web 2.0 Privacy and Security Workshop,Oakland, CA*, 2007.

[27] P. Hayton, B. Schölkopf, L. Tarassenko, and P. Anuzis. Support vector novelty detection applied to jet engine vibration spectra. In *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NIPS), Denver, CO*, pages 946–952, 2000.

[28] Z. He, X. Xu, J. Z. Huang, and S. Deng. FP-Outlier: Frequent pattern based outlier detection. *Computer Science and Information Systems*, 2(1):103–118, 2005.

[29] H. Heikinheimo, J. Vreeken, A. Siebes, and H. Mannila. Low-entropy set selection. In *Proceedings of the 9th SIAM International Conference on Data Mining (SDM), Sparks, NV*, pages 569–579. SIAM, 2009.

[30] K. Hempstalk, E. Frank, and I. Witten. One-class classification by combining density and class probability estimation. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium*, pages 505–519, 2008.

[31] S. Ho, Z. Lukacs, G. F. Hoffmann, M. Lindner, and T. Wetter. Feature construction can improve diagnostic criteria for high-dimensional metabolic data in newborn screening for medium-chain acyl-CoA dehydrogenase deficiency. *Clinical Chemistry*, 53(7):1330–1337, 2007.

[32] M. I. Jordan. Lecture 5: Properties of kernels and the gaussian kernel. CS281B/Stat241B: Advanced Topics in Learning & Decision Making, April 2004.

[33] P. Juszczak, N. M. Adams, D. J. Hand, C. Whitrow, and D. J. Weston. Off-the-peg and bespoke classifiers for fraud detection. *Computational Statistics & Data Analysis*, 52(9):4521–4532, 2008.

[34] H. Kantz and T. Schreiber. *Nonlinear Time Series Analysis*. Cambridge University Press, 2003.

[35] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi. He says, she says: conflict and coordination in Wikipedia. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), San Jose, CA*, pages 453–462, 2007.

[36] A. Knobbe and E. Ho. Pattern teams. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany*, pages 577–584, 2006.

[37] K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM), Columbus, OH*, pages 153–164, 2010.

[38] P. Laskov, C. Schäfer, and I. Kotenko. Intrusion detection in unlabeled data with quarter-sphere support-vector machines. In *Proceedings of 1st International Workshop on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), Dortmund, Germany*, pages 71–82, 2004.

[39] C.J. van Leeuwen and T.G. Vermeire. *Risk Assessment of Chemicals: An Introduction*. Springer, 2007.

[40] M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks item sets that matter. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany*, pages 585–592, 2006.

[41] M. van Leeuwen, J. Vreeken, and A. Siebes. Identifying the components. *Data Mining and Knowledge Discovery*, 19(2):173–292, 2009.

[42] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.

[43] Y. Liu, B. Cukic, and S. Gururajan. Validating a neural network-based on-line adaptive system. Technical Report Verification and Validation of Adaptive Systems, West Virginia University, 2006.

[44] G. Loosli, S.-G. Lee, and S. Canu. Context changes detection by one-class SVMs. In *Proceedings Workshop on Machine Learning for User Modeling: Challenges (UM), Edinburgh, Scotland*, pages 27–34, 2005.

[45] J. Ma and S. Perkins. Online novelty detection on temporal sequences. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC*, pages 613–618, 2003.

[46] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN), Portland, OR*, pages 1741–1745, 2003.

[47] B. Mak, J. T. Kwok, and S. Ho. Kernel eigenvoice speaker adaptation. *IEEE Transactions on Speech and Audio Processing*, 13(5):984–992, 2005.

[48] M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, pages 573–581, 2011.

[49] M. Markou and S. Singh. Novelty detection: a review. *Signal Processing*, 83(12):2481–2521, 2003.

[50] S. Marsland. Novelty detection in learning systems. *Neural Computing Surveys*, 3(2):157–195, 2003.

[51] A. K. McCallum. Bow: a Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering. Software available at `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[52] T. Mielikäinen and H. Mannila. The pattern ordering problem. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Cavtat-Dubrovnik, Croatia*, pages 327–338, 2003.

[53] A.J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P.J. H. Reijnders, F. Spitzenberger, M. Stubbe, J.B.M. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.

[54] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 2(2):181–201, 2001.

[55] S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.

[56] K. Narita and H. Kitagawa. Outlier detection for transaction databases using association rules. In *Proceedings of the 9th International Conference on Web-Age Information Management (WAIM), Zhangjiajie, China*, pages 373–380, 2008.

[57] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

[58] M. Nisenson, I. Yariv, R. El-Yaniv, and R. Meir. Towards behaviometric security systems: Learning to identify a typist. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Cavtat-Dubrovnik, Croatia*, pages 363–374, 2003.

[59] NIST/SEMATECH. *e-Handbook of Statistical Methods*. 2003.

[60] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Physical Review Letters*, 45(9):712–716, September 1980.

[61] M. Potthast. Crowdsourcing a Wikipedia vandalism corpus. In *Proceedings of the 33rd International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 789–790, 2010.

[62] M. Potthast and R. Gerling. Wikipedia Vandalism Corpus Webis-WVC-07. `http://www.uni-weimar.de/medien/webis/research/corpora`, 2007.

[63] M. Potthast, B. Stein, and R. Gerling. Automatic vandalism detection in Wikipedia. In *Proceedings of the 30th European Conference on IR Research (ECIR), Glasgow, Scotland*, pages 663–668, 2008.

[64] R. Priedhorsky, J. Chen, S. T. K. Lam, K. Panciera, L. Terveen, and J. Riedl. Creating, destroying, and restoring value in Wikipedia. In *Proceedings of the International ACM Conference on Supporting Group Work (GROUP), Sanibel Island, FL*, pages 259–268, 2007.

[65] L. Rassbach, T. Pincock, and B. Mingus. Exploring the feasibility of automatically rating online article quality. In *Proceedings of the International Wikimedia Conference (Wikimania), Taipei, Taiwan*, 2007.

[66] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[67] B. Schölkopf and A. Smola. *Learning with Kernels : Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[68] D. Sculley and C. E. Brodley. Compression and machine learning: a new perspective on feature space vectors. In *Proceedings of the Data Compression Conference (DCC), Snowbird, UT*, pages 332–341, 2006.

[69] A. Siebes and R. Kersten. A structure function for transaction data. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 558–569, 2011.

[70] A. Siebes, J. Vreeken, and M. van Leeuwen. Item sets that compress. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM), Bethesda, MD*, pages 393–404, 2006.

[71] K. Smets, B. Goethals, and B. Verdonk. Automatic vandalism detection in Wikipedia: Towards a machine learning approach. In *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy (WikiAI), Chicago, IL*, pages 43–48, 2008.

[72] K. Smets, B. Verdonk, and E. M. Jordaan. Discovering novelty in spatio/temporal data using one-class support vector machines. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks (IJCNN), Atlanta, GA*, pages 2956–2963, 2009.

[73] K. Smets and J. Vreeken. The odd one out: Identifying and characterising anomalies. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM), Mesa, AZ*, pages 804–815, 2011.

[74] K. Smets and J. Vreeken. SLIM: Directly mining descriptive patterns. In *Proceedings of the 12th SIAM International Conference on Data Mining (SDM), Anaheim, CA*, 2012.

[75] P. Sun and S. Chawla. On local spatial outliers. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM), Brighton, UK*, pages 209–216, 2004.

[76] N. Tatti and J. Vreeken. Finding good itemsets by packing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy*, pages 588–597, 2008.

[77] D.M.J. Tax. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, 2001.

[78] D.M.J. Tax. DDtools, the data description toolbox for Matlab, 2011. Software available at `http://prlab.tudelft.nl/david-tax/dd_tools.html`.

[79] D.M.J. Tax and R.P.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[80] C. Tong and V. Svetnik. Novelty detection in mass spectral data using support vector machine method. In *Proceedings of the 34th Symposium on the Interface (INTERFACE), Montréal, Canada*, 2002.

[81] T. Van den Bulcke, P.V. Broucke, V.V. Hoof, K. Wouters, S.V. Broucke, G. Smits, E. Smits, S. Proesmans, T.V. Genechten, and F. Eyskens. Data mining methods for classification of Medium-Chain Acyl-CoA dehydrogenase deficiency (MCADD) using non-derivatized tandem MS neonatal screening data. *Journal of Biomedical Informatics*, 44(2):319–325, 2011.

[82] S. Van Vaerenbergh, E. Estebanez, and I. Santamaria. A spectral clustering algorithm for decoding fast time-varying BPSK MIMO. In *Proceedings of the 15th European Signal Processing Conference (EUPISCO), Poznan, Poland*, 2007.

[83] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[84] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), Vienna, Austria*, pages 575–582, 2004.

[85] J. Vreeken and A. Siebes. Filling in the blanks – KRIMP minimisation for missing data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy*, pages 1067–1072, 2008.

[86] J. Vreeken, M. van Leeuwen, and A. Siebes. Characterising the difference. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Jose, CA*, pages 765–774, 2007.

[87] J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[88] C. S. Wallace. *Statistical and Inductive Inference by Minimum Message Length*. Springer-Verlag, 2005.

[89] C. Wang and S. Parthasarathy. Summarizing itemset patterns using probabilistic models. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, pages 730–735, 2006.

[90] G. I. Webb. Discovering significant patterns. *Machine Learning*, 68(1):1–33, 2007.

[91] G. I. Webb. Self-sufficient itemsets: An approach to screening potentially interesting associations between items. *ACM Transactions on Knowledge Discovery from Data*, 4(1):1–20, 2010.

[92] S. Wold, P. Geladi, K. Esbensen, and J. Öhman. Multi-way principal components and PLS-analysis. *Journal of Chemometrics*, 1(1):41–56, 1987.

[93] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*, pages 314–323, 2005.

[94] C. Yuan, C. Neubauer, Z. Cataltepe, and H.-G. Brummel. Support vector methods and use of hidden variables for power plant monitoring. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Philadelphia, PA*, pages 693–696, 2005.

Samenvatting

De toegenomen opslag- en verwerkingscapaciteit maken het verzamelen en bewaren van gegevens steeds makkelijker. Er is echter een heel proces nodig om kennis te verwerven uit deze gegevens of om deze gegevens in een toepassing te gebruiken. Eén van de pijlers binnen dit proces is datamining. In dit domein worden algoritmen ontwikkeld om enerzijds de gegevens samen te vatten in zogenaamde modellen, en anderzijds om onverwachte patronen of relaties te vinden in de gegevens. Om vat te krijgen op de steeds groeiende hoeveelheid aan gegevens, willen we dat de modellen en patronen zowel bruikbaar als verstaanbaar zijn voor de gegevensbezitter.

In dit proefschrift ontwikkelen we datamining technieken om, vertrekkende van de beschikbare gegevens met beperkte menselijke inspanning, modellen op te stellen met als doel onregelmatigheden, observaties afwijkend van de verwachte norm, in (nieuwe) gegevens accuraat te identificeren en begrijpelijk te karakteriseren.

Daar we in de praktijk geconfronteerd worden met een diversiteit aan mogelijke toepassingen en verschillende soorten gegevens, is het ontwikkelen van een allesomvattende methode waarbij in elke stap van het proces alle eisen tegelijk ingewilligd, geoptimaliseerd en gevalideerd worden een brug te ver. Doorheen dit proefschrift belichten we dan ook telkens enkele deelaspecten van deze probleemstelling. We gaan er hierbij telkens van uit dat, om het normmodel uit de gegevens op te bouwen, we voornamelijk beschikken over voorbeelden van de te verwachten situatie.

Na een korte algemene inleiding in hoofdstuk 1, gaan we in hoofdstuk 2 en 3 uit van specifieke praktische problemen. In hoofdstuk 2 stellen we een model op om onregelmatigheden te detecteren tijdens het opvolgen van productie processen in een chemische fabriek. In hoofdstuk 3 geven we een aanzet om vandalenstreken op te merken in Wikipedia.

Het aanduiden van deze onregelmatigheden is echter niet voldoende. We wensen een beschrijving die toelicht waarom bepaalde gegevens als onverwacht worden bestempeld. Door expliciet gebruik te maken van modellen die zijn opgebouwd met een beperkt aantal patronen die de normale verwachtingen beschrijven en de verschillen met de huidige observatie te belichten, verlenen we dit nodige inzicht in hoofdstuk 4. Het zonder tussenstap efficiënt vergaren van de compacte modellen vormt het onderwerp in hoofdstuk 5.