TU Delft

Master Thesis

Rule Induction on Multiple Instance Learning Concepts

Author:
Robin van der Wal

Supervisor:
Dr. DMJ Tax

A thesis submitted in fulfillment of the requirements for the degree of Master of Computer Science

in the

Pattern Recognition & Bioinformatics
Pattern Recognition Lab

October 16, 2022

TU DELFT

Abstract

EEMCS
Pattern Recognition Lab

Master of Computer Science

Rule Induction on Multiple Instance Learning Concepts

by Robin VAN DER WAL

Thesis supervisor:	Dr. DMJ Tax
Second reader:	Dr. H Wang
Thesis advisor:	Prof.Dr.Ir. MJT Reinders

Multiple Instance Learning (MIL) is a type of semi-supervised machine learning used recently in medical and multi-media fields. In MIL, instead of a single feature vector, a set of feature vectors has to be classified. Standard MIL algorithms assume that only some of these vectors are useful for building a classifier. This paper extends the standard MIL assumption by combining propositional logic and classical MIL classifiers. Adding propositional logic allows for increased interpretability as it establishes an if-then relationship between the input data and the output classes. This combination of logic and classical MIL classifiers will be called Concept Rule Induction (CRI). CRI is tested on several artificial and real-life bird song data. CRI is shown to work for these data sets, and the rules produced by propositional logic can be interpreted.

Acknowledgements

I want to thank my supervisor Dr. David MJ Tax, for all his advice and patience during the thesis period. David came up with the data set, and while the thesis took more effort than expected, in the end, I gladly submitted this thesis. I would also like to thank the people at my work in the advanced analytics and big data team at KPMG NL for their encouragement and for bringing structure to my life when I needed it the most. Lastly, I thank my family and friends for their support over the last few weird years during the pandemic.

Contents

List of Figures

Chapter 1

Introduction

Multiple Instance Learning (MIL) is a form of semi-supervised machine learning [4]. In 2008, Babenko gave a simple example [5] of a MIL problem. Imagine that there is a set of key chains, each with multiple keys, and the objective is to open a door with one of the key chains. The goal of a MIL algorithm is to identify the correct key chains that can open the door. In order to solve this problem, a MIL algorithm has to find the set of all keys that can open the door, while only knowing which *key chains* open the door, and which ones do not

In recent years, MIL algorithms have been created for all kinds of tasks. The notion of MIL is often employed for medical use cases such as tissue segmentation and cancer classification [6, 7, 8, 9, 10, 11]. Here, images are split into a set of feature vectors, in different ways such as segmentation or taking patches. MIL algorithms try to identify whether an image contains diseased tissues.

MIL algorithms have even been used in the fight against COVID-19 [12, 13, 14]. Furthermore, MIL algorithms can be employed for video and sound data [4, 15, 16, 17]. MIL has been used in this space for example for deep fake detection [18, 19].

These new MIL algorithms work well to find which key chains can open the door, as listed in the earlier example, however more and more new MIL algorithms make use of deep learning [10, 11, 12, 16]. Despite deep learning success in the last decade, it is often criticized for its black box approach [20]. Deep learning is mainly data-driven, causing unpredictable and incomprehensible results [21, 22]. There has been work on the understandability of deep networks, but these methods mainly work on image-based deep neural networks such as convolutional neural networks [23, 24, 25, 26].

One way of improving interpretability in machine learning is to make use of rule induction [27]. Rule induction has been used for MIL-like data sets before. For example, in the year 2000, the first basic rule induction algorithm was published [28, 29] to solve a mutagenicity (cancer-causing molecules) MIL classification problem. However, while the data was MIL-like, the papers did not use any notions of MIL [4].

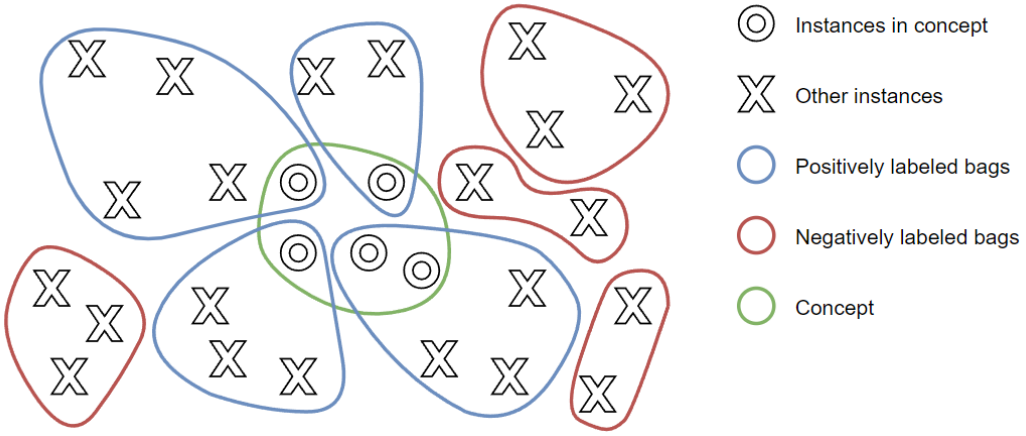
Another way to tackle understandability is the use of lazy learning methods, where new data is compared to a training set. These algorithms are called BayesianKNN and Citation-kNN and [4, 30]. These algorithms can give back the k closest examples in the training set. But they do not use any underlying structure (except for a different metric) for performance. In the key-chain example, these algorithms can give you back the closest k key-chains to the key-chain that is being examined. However, these algorithms will not provide the answer to whether an individual key opens a door or give reasons why.

Instead of relying on another deep learning approach, this paper will go back to the roots of MIL [4, 5]. In MIL nomenclature there is a well-known notion called the

concept. This paper will use this notion in combination with rule induction with the main goal of interpretability. The next section will introduce the general MIL setup in combination with concepts.

1.1 Multiple Instance Learning Concepts

An example of a toy multiple instance dataset can be seen in Figure 1.1. In this figure, there are data points with two features. These data points are called instances in MIL literature because these points are grouped together in sets, like the key chains. The groups are shown by the blue and red circles. These groups of instances are called bags in MIL literature. The bags are labeled and instances individually do not have a label. In this example, there are two classes, the positive class in blue and the negative class in red. So there are 4 negative labeled examples and 4 positive labeled examples. The number of instances per bag can differ, in this case between 2 and 5 instances.



`Figure 1.1:` An example of a two-dimensional Multiple Instance Problem. There are 4 bags for both classes. The positively labeled blue bags all have an instance in the center, the negatively labeled bags do not. The center here can be seen as the concept, which, when known, can be used to label new bags.

If the objective is to classify the positive bags from the negative bags in Figure 1.1, then you could look at what properties these positive bags have in common. These properties should then not be shared with negative bags. In the figure, the positive bags are all in the middle. So a possible classifier could check a bag for instance within a region, which is indicated in green. This is the so-called *concept* that defines the positive class and what distinguishes it from the negative class [4]. In the example of the key chains, the concept would be keys that fit the lock of the door.

The assumption that a single instance is enough to correctly classify a bag is sometimes not enough. Take the example described in [4], where a traffic controller wants to automatically detect traffic jams from images from roads. If this problem is tackled as a MIL problem, the objects in the images are instances and a bag is an image from a road. It is clear that a car instance would contribute to traffic jams, but a single car isn't a traffic jam by itself. So detecting the concept of a car would not be enough, but detecting this concept multiple times in combination with a threshold can detect a traffic jam.

The summation of the same concept in combination with a threshold is still not enough to classify all MIL problems. A clear example of the combination of multiple concepts is given by Foulds and Frank [31]. Here the authors want to classify images into three categories: desert images, ocean images, and beach images. Images are segmented into multiple instances. A desert image should contain a sand segment, an ocean image should contain a water segment and a beach image has to contain both the water and the sand segment.

The combination of MIL *concepts* has been used before in MIL algorithms. One generally well-performing MIL algorithm that does this is called MILES [4, 32]. This algorithm produces linear weighted combinations of instances to classify MIL data. This paper will show that there is a toy problem that can not be solved with just linear combinations.

1.2 Research question

This paper will use a rule induction algorithm on MIL concepts to classify MIL problems [33]. This new algorithm will be called Concept Rule Induction (CRI). This paper will look at how to produce concepts and rules from two existing techniques: MILES and k-means [30, 32]. Then to tune CRI and understand its behavior, several toy problems are created specifically for testing combined MIL concepts. Next, a real-life audio data set of birds singing is used to examine CRI, with the goal of extracting sound fragments of specific bird species from a longer mixed species sound fragment.

> *RQ 1: Is it possible to extend bag concept classifiers with propositional logic to create a MIL classifier, with improved interpretability?*

To help answer this question, this paper will dive into four sub-questions. Firstly, the MIL exclusive OR problem is an example of a non-linear combination of concepts, and solving this problem is a starting point for CRI. This is in combination with the question of how to tune concept detection algorithms as, without good hyper-parameters, the concepts cannot be found. After finding a suitable concept detection algorithm, CRI can be used to produce rules. To interpret these rules, a toy data set containing obvious rules can be used, here CRI should produce rules that are contained in the toy data set. Lastly, CRI must also perform on a real-life data set, to show that it can be used in real-life machine-learning settings. The sub-questions are listed below explicitly.

> *RQ 1.1: Can simplified propositional logic in a bag classifier solve the MIL XOR problem?*
>
> *RQ 1.2: What is a usable concept classifier and how should it be tuned?*
>
> *RQ 1.3: Can propositional logic on a concept classifier be used to generate meaningful rules on data that contains structure?*
>
> *RQ 1.4: Is it possible to use the concept assumption extension in a real data set?*

To start, the foundation of MIL and rules is described in chapter 2. Next, CRI itself is proposed with two possible methods for concept detection, MILES and k-means, and one method for rule induction, Prism, in chapter 3. The two methods for rule induction are compared on several toy problems in section chapter 4, solving RQ

1.1 and RQ 1.2. Then after choosing a concept detection method, two new toy data sets based on the MNIST data set are used to examine the rules produced by CRI in chapter 5, solving RQ 1.3. This will give some insight into how to use the produced rules. Then the information in both chapter 4 and chapter 5 will be combined to use CRI on bird song real-life data set in chapter 6, solving RQ 1.4. Lastly. the conclusion on RQ 1 is made in chapter 7, together with a discussion on improvements on CRI.

Chapter 2

Multiple Instance Learning

This section introduces Multiple Instance Learning (MIL) and some additional *concepts* which are required for the new method of this paper. The first section zooms in from basic machine learning to MIL. The first section can be skipped if the reader is already well-versed in MIL. The next section will go over assumptions researchers make on multi-instance data and how that can be extended. The last section will discuss propositional logic used in a MIL setting.

2.1 From machine learning to multiple instance learning

Learning can be defined as the process where one increases their problem-solving ability through experience [34]. In the age of computerized automation, machine learning has been central in recent years. For this, researchers have been designing algorithms for decades and many different sub-fields have sprung up [35, 36]. The basis of machine learning is that experience is provided by training examples

Machine learning can be split into unsupervised and supervised algorithms. Supervised algorithms are the most common set of machine learning algorithms. The experience that the machine will consume to learn is labeled data. The data consist of a set of feature vectors, which each describe a single data point in the data set. Each data point has a label that describes its class. To abstract classification, the goal is to find a function f which can predict a class y [37] on an object. If an object has d features and there are Q classes, then describe f as

$$f : \mathbb{R}^d \to \{y_1, .., y_Q\}. \tag{2.1}$$

However, labeling costs resources, often requiring experts to look at the data. There have been attempts to mitigate this, for example crowd-sourcing the effort using Amazon's Mechanical Turk [38], but this approach comes with its own problems such as label accuracy and missing expert feedback [39]. To still use data that is unlabeled, researchers have developed unsupervised techniques. There are plenty of examples of unsupervised machine learning. Most clustering algorithms (such as k-means [40]) don't require any labeled data and use the structure of the data to learn patterns [41].

Between total unsupervised learning and supervised learning there is a field called semi-supervised learning [42]. In semi-supervised learning, only part of the data is labeled, and often a large part is not. The labeled data alone is not enough to build a robust machine learning algorithm but, with the help of assumptions on structure within the dataset, it can be possible to employ the unlabeled data as well. This is the domain where Multiple Instance Learning fits.

Multiple Instance Learning (MIL) is a form of semi-supervised learning [43]. In the case of MIL, instead of a single labeled data point, there can be multiple data points $\boldsymbol{x} \in \mathbb{R}^d$ that have a single class y_q. This collection is called a bag $\boldsymbol{B}$. The notation for the jth instance in the ith bag in a set of bags is $\boldsymbol{x}_{ij} \in \boldsymbol{B}_i$.

2.2 Multiple instance assumptions

In MIL classification only the bags are labeled and no additional label information on the instances is given. One of the goals of MIL is to be able to classify new bags based on a training set of other bags. Such an algorithm is called a bag classifier, which can be written as $g(\boldsymbol{B}_i) \rightarrow \{y_1, .., y_Q\}$. In order to create such a classifier, researchers made assumptions about MIL data structures [4, 31, 43, 44]. This subsection will introduce these assumptions and show examples of them.

The first assumption was introduced in 1998 [43], here was assumed that instances are independent of each other, but there are some instances that do individually contribute to the label. This assumption is still used as the starting point for papers recently [45, 46].

From now on, only the two-class classification problem will be considered, where there is a positive or negative class. A single instance that belongs to a positive class will make the bag labeled as positive. The two-class notation will simplify notation and can be extended back to a multi-class problem using a one-against-all approach. Because instances here are independent, an instance classifier can be used, such as f from Equation 2.1. This results in

$$g(\boldsymbol{B}_i) = \begin{cases} 1, & \exists \mathbf{x}_{ij} \in \boldsymbol{B}_i : f(\mathbf{x}_{ij}) = 1. \\ -1, & otherwise. \end{cases} \tag{2.2}$$

This assumption was based on the work of Dietterich et all [44]. In this paper, scientists tried to predict molecule drug activity. A single molecule can have multiple physical atom orientations based on its energy level. A different physical orientation results in different molecular properties. As a test, the authors tried to predict if a molecule would smell musky, based on all its physical configurations. As long a single configuration would smell musky, the molecule, as a bag, would be labeled positive.

As discussed in the introduction, later MIL papers used the notion of MIL concepts. If a single concept c exists that can separate the positive and negative class of a MIL problem, then the entire feature space $\mathbb{R}^d$ can be divided into instances that belong to the concept and those which do not.

$$f(\mathbf{x}_i) = \begin{cases} 1, & \mathbf{x}_{ij} \in c, \\ -1, & \mathbf{x}_{ij} \notin c. \end{cases} \tag{2.3}$$

In the example of the musky molecules, a concept would be an example of a physical molecule configuration that smells musky.

Equation 2.3 is enough to describe the key chain example from the introduction, but slightly more notation is needed to formalize the other examples. Let's continue with the traffic jam example. Here, a single car in the image was not enough to count as a traffic jam. Let f_{car} be an instance classifier for cars, and add a threshold θ which indicates enough cars for a traffic jam. Then construct a MIL classifier for traffic jams

as follows:

$$g(\boldsymbol{B}_i) = \begin{cases} 1, & \sum_{\mathbf{x}_{ij} \in \boldsymbol{B}_i} f_{\text{car}}(\boldsymbol{x}_{ij}) \geq \theta, \\ -1, & otherwise, \end{cases} \tag{2.4}$$

with $\boldsymbol{B}_i$ the traffic image as a MIL bag. This MIL bag classifier $g(\boldsymbol{B}_i)$ allows for the counting of concepts. Now, to include examples of deserts, oceans, and beaches from the introduction, it is needed to have more than a single concept [47]. Let $\mathcal{C}$ be the collection of concepts for a specific class and let f_c be the instance classifier for concept c, then Equation 2.4 can be extended as such:

$$g(\boldsymbol{B}_i) = \begin{cases} 1, & \forall c \in \mathcal{C} : \sum_{\mathbf{x}_{ij} \in \boldsymbol{B}_i} f_c(\boldsymbol{x}_{ij}) \geq \theta_c, \\ -1, & otherwise. \end{cases} \tag{2.5}$$

In the case of the beach class, the set of concepts would be $\mathcal{C}_{\text{ocean}} = \{c_{water}, c_{\text{sand}}\}$. With this bag classifier possible to move on to the use of rule induction for MIL classification on concepts.

2.3 Concepts to rules

This section will go over the process to create if-then rules based on the extracted concepts. To help with notation, this section will also introduce propositional logic. Propositional logic is a set of formal definitions of axioms, where the goal is to evaluate the truthfulness of specific statements [33].

With Equation 2.5, beaches can be classified with the two concept classifiers f_{sand} and f_{water}. However, the other two classes of ocean and desert cannot be classified directly with the same concept classifiers. For example, take the ocean class. It is possible to take the water classifier f_{water}, but an image of a beach would still classify as an ocean because it contains the water concept. Here it would be better to use a negative classifier for sand, $f_{\neg\text{sand}} = \neg f_{\text{sand}}$, and combine it with the water classifier. Then introduce the *negation* operator ($\neg$) and the *and* operator ($\wedge$) to combine these concepts into propositional formulas.

A propositional formula for ocean could be $r_{\text{ocean}} = f_{\text{water}} \wedge \neg f_{\text{sand}}$, which act as a *rule*. This rule can then be tested against all instances of a bag $\boldsymbol{B}_i$ and check for both concepts if there is an instance $\boldsymbol{x}_{ij}$ for which $f_{water}(\boldsymbol{x}_{ij}) = 1$ and no instance $\boldsymbol{x}_{iz}$ for which $f_{sand}(\boldsymbol{x}_{iz}) = 1$. If this rule is true for bag $\boldsymbol{B}_i$, then the bag $\boldsymbol{B}_i$ can be classified as a picture containing an ocean.

If the MIL problem needs more than a single rule made out of just concept classifiers, negation operators, and *and* operators, then these rules can be combined with the *or* operator $\vee$. Let the set of all rules for a specific class $y \in Q$ a rule set be defined as $\boldsymbol{R}_y$ and evaluate a bag $\boldsymbol{B}_i$ by writing $r_k(\boldsymbol{B}_i)$, then predict and test for class y as

$$\boldsymbol{R}_y \rightarrow \hat{y} \iff \exists\, r_k \in \boldsymbol{R}_y : r_k(\boldsymbol{B}_i) = True. \tag{2.6}$$

This can be brought back to the bag classifier g_y for class y as follows:

$$g_y(\boldsymbol{B}_i) = \begin{cases} 1, & \exists\, r_k \in \boldsymbol{R}_y \ : \ r_k(\boldsymbol{B}_i) = True, \\ -1, & otherwise. \end{cases} \tag{2.7}$$

While Equation 2.5 can be inverted for a simple case such as the non-beach classifier described above, it is not enough in case multiple concepts combinations such as the *eXclusive OR* (XOR) problem. In the XOR problem, there are two concepts A

and B, and the positive class is an XOR operator applied to these two concepts. So the rule set would consist of two rules:

$$\boldsymbol{R}_{\text{XOR}} = \begin{cases} r_1 : f_A \wedge \neg f_B, \\ r_2 : \neg f_A \wedge f_B. \end{cases} \tag{2.8}$$

With the extension Equation 2.7 it is possible to create a bag classifier that can handle the XOR problem of Equation 2.8. This method of combining concept classifiers with rule induction will be called Concept rule Induction and is described in chapter 3.

Chapter 3

Method: Concept Rule Induction

This section will describe the proposed MIL classifier. This new method combines a concept finder algorithm with a rule induction algorithm to create a bag classifier. The method will be called Concept Rule Induction (CRI). Because CRI combines two existing methods, this section will go over both the concept finder algorithm and the rule inductions methods. After that, this section will also list the assumptions made for the classification approach.

CRI will create a bag classifier using propositional logic described in section 2.3, by first creating a bag-level feature based on concepts. This first part aims to find a set of concepts $\mathcal{C}$ which can help discriminate between two classes and then check for each bag if these concepts occur. This is similar to Equation 2.5, but the concept classifier f_c will be used on the entire bag $\boldsymbol{B}_i$. These concept classifiers f_c will be binary, with a 1 indicating the presence of a concept c and a 0 indicating the absence of c. If a concept detection method finds N_c concepts, then for each bag, a binary vector can be created as follows:

$$\boldsymbol{f}_c(\boldsymbol{B}_i) = [f_1(\boldsymbol{B}_i), f_2(\boldsymbol{B}_i), \ldots, f_{N_c}(\boldsymbol{B}_i)]\,. \tag{3.1}$$

Starting from a MIL training data set $\boldsymbol{B}$, the concepts can be extracted using an existing MIL algorithm or a clustering algorithm. The data set $\boldsymbol{B}$ can then be converted to a bag-level feature data set, which will be named the bag concept matrix $\boldsymbol{C_B}$. If there are $N_{\boldsymbol{B}}$ training bags and there are N_C concepts found, then $\boldsymbol{C_B}$ looks like:

$$\boldsymbol{C_B} = \begin{bmatrix} \boldsymbol{f}_c(\boldsymbol{B}_1) \\ \vdots \\ \boldsymbol{f}_c(\boldsymbol{B}_{N_{\boldsymbol{B}}}) \end{bmatrix} = \begin{bmatrix} f_1\left(\boldsymbol{B}_1\right) & \ldots & f_{N_C}\left(\boldsymbol{B}_1\right) \\ \vdots & \ddots & \vdots \\ f_1\left(\boldsymbol{B}_{N_{\boldsymbol{B}}}\right) & \ldots & f_{N_C}\left(\boldsymbol{B}_{N_{\boldsymbol{B}}}\right) \end{bmatrix}. \tag{3.2}$$

The bag concept matrix $\boldsymbol{C_B}$ represents the MIL training data B as a more regular machine learning problem. Each row can be seen as a binary bag-level feature factor and has a class label. The bag concept matrix can then be fed into rule induction algorithms [48]. The rule inducting will extract the rule sets $\boldsymbol{R}$ per class by using a one-versus-all approach. This approach is taken so that even in a multi-class setting, the more straightforward situation of a positive and negative class can be used. CRI is visualized in Figure 3.1.

Now that the outline is defined, the rest of this chapter will examine concept classifiers and then the rule induction method used in this paper. In the end, this section will discuss some assumptions made that have been made for the use of CRI and how both components of CRI will be examined.

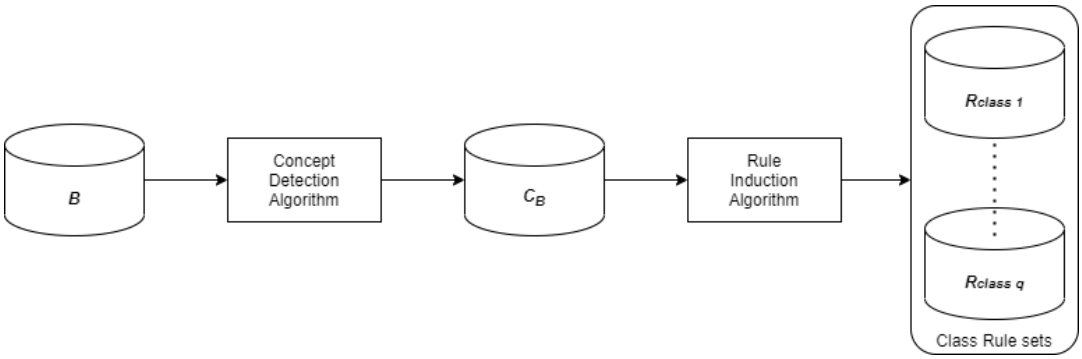


Figure 3.1: Schematic of Concept Rule induction for Equation 2.7. $\boldsymbol{B}$ is a MIL data set. $\boldsymbol{C_B}$ is the bag concept matrix and rules sets $\boldsymbol{R}$ are split up per class.

3.1 Concept detection algorithms

There are multiple concept-based algorithms in the multiple instance domain [4]. This paper will look at two candidates: a common MIL algorithm called MILES and k-means. The MILES algorithm is a staple tool to compare new MIL algorithms against [4], and it has a built-in feature that can be used as a concept classifier. The k-means approach, on the other hand, is a simple way to do clustering on a training data set to identify clusters in the training space, which might be useful as concepts. This section will show how to construct the bag concept matrix $\boldsymbol{C_B}$ from each method and their differences.

3.1.1 Multiple Instance Learning via Embedded Instance Selection

One of the most successful MIL concept-based algorithms is MILES [4]. MILES stands for Multiple Instance Learning via Embedded Instance Selection and was devised in 2006 by Y. Chen et all. [32]. This section will describe the key details of MILES, but additional information on MILES and how it was used in this paper can be found in Appendix A.

As its name implies, MILES chooses instances that act as potential concepts by embedding MIL bag data using the instance data. The embedding is done by using *most-likely-cause* estimator from the diverse density framework [49]. This estimator $s(\boldsymbol{x}^l, \boldsymbol{B}_i)$ is an indicator how far a training instance $\boldsymbol{x}^l$ is from a bag $\boldsymbol{B}_i$. The most-likely-cause estimator is defined as

$$s(\boldsymbol{x}^l, \boldsymbol{B}_i) = \max_j \exp\left(-\frac{||\boldsymbol{x}_{ij} - \boldsymbol{x}^l||^2}{\sigma^2}\right), \tag{3.3}$$

where σ is the scaling factor that depends on the training data distribution. This embedding is calculated for all bags and all their instances in the training data set. After which a *sparse* linear classifier is used to select instances which act as concepts [50].

The classification $\hat{y}_i$ on a bag $\boldsymbol{B}_i$ is defined as:

$$\hat{y}_i = \text{sign}\left(\sum_{l=1}^{N_l} w_l^* s(\boldsymbol{x}_l, \boldsymbol{B}_i) + b^*\right), \tag{3.4}$$

where N_l is the number of instances in the training set $\boldsymbol{B}$, $s(\boldsymbol{x}_l, \boldsymbol{B}_i)$ the embedding of bag $\boldsymbol{B}_i$ on instance $\boldsymbol{x}_l$ from the training set $\boldsymbol{B}$ and w_l^* together with b^* are the learned weights of the sparse linear classifier. In Appendix A, there is an explanation on how to transform Equation 3.4 to a linear program that acts as sparse linear classifier. The sparse linear classifier introduces an additional parameter. This parameter is called the regularisation parameter λ, and during optimization, it forces the number of non-zero weights down.

The learned weights can be used to create a concept classifier like that of Equation 3.1. The linear classifier is sparse, so most of the learned weights w_l^* should be zero. Select the training instances $\boldsymbol{x}_c$ for which the corresponding weight is non-zero $|w_c^*| > 0$ to act as the center point of a concept. The *most-likely-cause* uses a parameter σ as Gaussian weighting on the training instances. This parameter σ can be compared to the distance between the non-zero instances $\boldsymbol{x}_c$ and the instances from the new bag $\boldsymbol{B}_i$. A single concept classifier f_c is constructed in the following formula:

$$f_c\left(\boldsymbol{B}_i\right) = \begin{cases} 1 & \exists\, \boldsymbol{x}_{ij} \in \boldsymbol{B}_i : \|\boldsymbol{x}_c - \boldsymbol{x}_{ij}\| < \sigma, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

All the non-zero learned weights from Equation 3.4 are used in Equation 3.5 to create the bag concept matrix $\boldsymbol{C_B}$ from Equation 3.2. Each row in this matrix is a different bag $\boldsymbol{B}_i \in \boldsymbol{B}$ from the training data. Each column represents a different concept c from $|w_c^*| > 0$. And the individual matrix entries are the result of using the concept classifier f_c on bag $\boldsymbol{B}_i$.

3.1.2 k-Means clustering

Clustering algorithms group together clusters of data points. In MIL, bags contain multiple instances, but these can be anywhere in the feature space. Using a clustering algorithm on just the instances will allow for instances in a bag to be assigned to multiple clusters. As a result, a cluster of instances could act as a concept. Each bag can then get a binary indicator for which clusters it contains, making it possible to create the bag concept matrix $\boldsymbol{C_B}$ from Equation 3.2. Plenty of clustering algorithms exist, but to test CRI, a simple clustering algorithm would be adequate, such as k-means [51].

k-Means is a well-known and relatively old algorithm. The name was first used in a paper from 1956 [40]. k-Means initializes by taking k random cluster points in the feature space. Then for each training point in the space, the distance between it and the k cluster points is calculated, and each training point is assigned to its closest cluster. After this step, the mean position of each set of data points per cluster point is calculated, and this mean position is set to be the new cluster point. Repeat the previous steps until no new assignments of clusters are made. The data points are then divided into k clusters with k means.

To use k-means as a concept detector for a bag $\boldsymbol{B}_i$, let the k means act as concepts. Next, check for each instance $\boldsymbol{x}_{ij} \in \boldsymbol{B}_i$, which of the k mean clusters concept is the closest, then that mean cluster concept is present in bag $\boldsymbol{B}_i$. Because a bag $\boldsymbol{B}_i$ contains multiple instances, it can contain multiple k-means cluster concepts. The result is the bag concept matrix $\boldsymbol{C_B}$ from Equation 3.2, which can then be fed to a rule induction algorithm.

3.1.3 Why these concept detection algorithms

Both MILES and k-means are suitable candidates for concept detection in CRI. Both methods can produce the bag concept matrix $\boldsymbol{C_B}$ needed for the rule induction step. MILES itself is a staple MIL algorithm with great performance [4]. The instance embedding step from Equation 3.3 in combination with the sparse linear classifier used for Equation 3.4 creates a number of optimized concepts depending on the regularization parameter. And for k-means, the number of concepts is equal to the k parameter. This means that both methods can be fined tuned on toy problems that set the number of expected concepts. The result of CRI can then be examined by these toy problems, which is why MILES and k-means were chosen.

The difference between MILES and k-means is that they construct concepts differently. MILES uses a bag embedding step and relies more on the distances between these bags. k-Means looks only at the instance data, without taking any of the properties of MIL into account. Both methods are expected to perform differently depending on the input data, and examing where each method performs better will be the goal of chapter 4. For now, let's define the second part of CRI.

3.2 Rule induction by Prism

There are multiple rule induction algorithms that are suitable to process $\boldsymbol{C_B}$ [52]. However, to prove the validity of CRI, the most basic rule induction algorithm will be used. Depending on the use case, a more sophisticated rule induction method could be used in the future. The algorithm that will be used in this paper is the Prism algorithm [53]. This section will give a brief outline of the algorithm, but a more extended version can be found in Appendix B.

The Prism algorithm is a rule induction algorithm that can directly induce classification rules based on $\boldsymbol{C_B}$. It does this by selecting a combination of attribute-value pairs which maximize the coverage of the desired outcome class. This is also called a sequential covering algorithm. The attribute-value pairs in $\boldsymbol{C_B}$ are $f_c(\boldsymbol{x}_c, \boldsymbol{B}_t)$ and its negation $\neg f_c(\boldsymbol{x}_c, \boldsymbol{B}_t)$. In each step, Prism takes the best-ranking attribute-value pair and adds it to a rule. The rank of an attribute-value pair is determined by how many rows in $\boldsymbol{C_B}$ of the target class it covers and how many rows of other classes it excludes. If adding additional attribute-value pairs does not increase the coverage on the target class anymore, it will add the rule to the rule set $\boldsymbol{R}$. Any rows from $\boldsymbol{C_B}$ which are covered by the rule are removed. This is done until $\boldsymbol{C_B}$ no longer contains any rows which belong to the target class.

While this algorithm is simple in nature, it can easily be extended with different selecting and stopping criteria. For example, the adding of the attribute value pairs could be stopped earlier if the coverage on $\boldsymbol{C_B}$ does not increase significantly. This would help with overfitting the training set. There are also multiple attempts to fine-tune Prism rule induction on specific use cases over the years. Thus, if the base version of Prism works, then it can be improved on particular cases in the future [52, 54, 55].

3.3 Considerations on Concept Rule Induction

For CRI several assumptions are made:

1. *Concepts are separable.* Concepts are assumed to be separable, so the classification can identify the different concepts

2. *Rules are non-probablistic.* It is assumed that a combination of concepts can uniquely define a class.

When using CRI on real-life data, it should be checked whether these two assumptions hold. If not, one could use a different concept detector or rule induction method, but this is out of the scope of this paper. A consequence of these two assumptions is that concepts have hard cut-off points. If concepts overlap in feature/space, they are non-separable. If concepts are non-separable, CRI will not be able to find a concise rule set and its performance will deteriorate.

Now that both components are defined, they will be examined on how to configure them. In chapter 4, both MILES and k-means will be discussed on a simple toy problem to see how they perform and what the pros and cons of each method are. To follow that up, in chapter 5, the application of Prism on the bag concept matrix will be examined on how the rules can be interpreted and used in combination with the concepts. Then with information gained from those two sections, the application of CRI on a real-life data set will be tested in chapter 6.

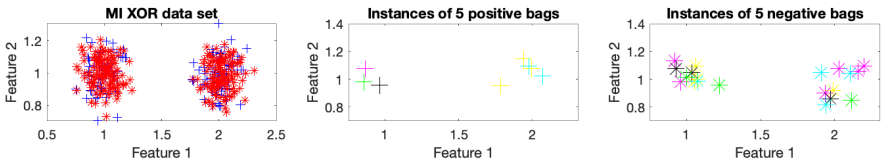
Chapter 4

Concept detection

To answer whether CRI can solve the MIL XOR problem, the concept classifiers MILES and k-means can be used with default settings. To decide which concept classifier is best, the methods have to be tuned well, which is discussed in Appendix D. For MILES, the scaling factor σ depends on the size of the expected concepts, and the regularization parameter λ should be set so that the optimizer can still find a solution. For k-means, k should be set to the number of expected concepts, and if not known, k should be increased till the performance of CRI plateaus. This section shows that CRI can solve the XOR problem, and then tackles three more difficult XOR-like problems to compare both concept detection algorithms.

4.1 MIL eXclusive OR toy problem

The MIL XOR toy problem is set up in the following way: there are two clusters, which are called concept A and concept B. The property of the positive class is that bags from it only contain instances in concept A or in concept B, but a positive bag never contains instances that appear in both concepts. The clusters were generated by two multi-variate distributions. The negatively labeled bags always have at least one instance in both concepts. An example of this data set is in Figure 4.1. This subsection will only examine MILES as a concept detector to show that CRI can solve the XOR problem because it's clear that k-means can find the correct cluster if $k = 2$.



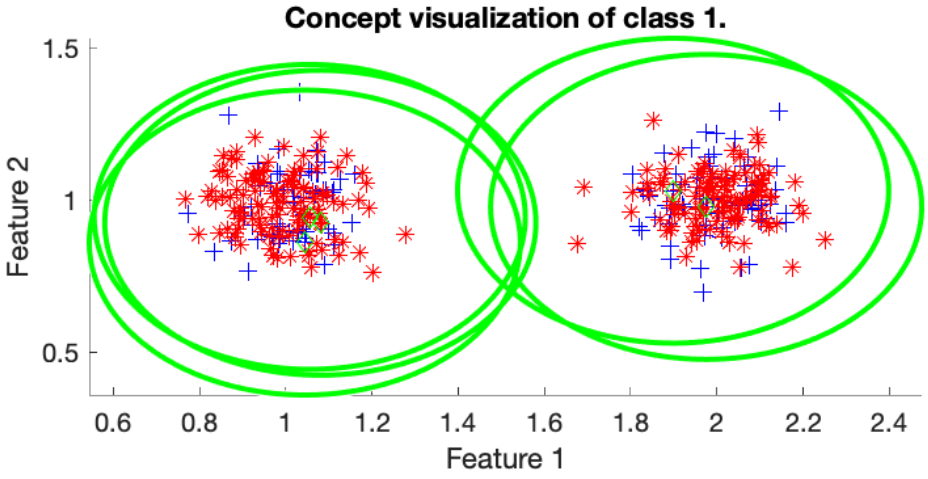
`Figure 4.1:` The Multiple Instance eXclusive OR data set consisting of fifty positive and negative bags. On the left: is the entire data set with positive bags in blue, and negative bags in red. In the middle are five positively labeled bags. On the right are five negatively labeled bags in different colors which are negative.

The concepts, or instances with non-zero weight from MILES, can be seen in Figure 4.2. These concepts were generated by using $\sigma = 0.5$ and $\lambda = 0.1$. More concepts are found that are needed, but further increasing the regularization parameter λ would not generate a valid solution with the linear program. The Prism algorithm can reduce the number of concepts needed for the final bag classifier. After running Prism

the following rule set is retrieved:

$$\boldsymbol{R}_{\text{XOR}} = \begin{cases} r_1 : \neg f_3, \\ r_2 : \neg f_1. \end{cases}$$

At first glance, $\boldsymbol{R}_{\text{XOR}}$ seems sufficient to classify the positive class. r_1 contains $\neg f_3$ which is a concept classification function around concept A and r_2 contains $\neg f_1$, which comes from concept B. Thus, $\boldsymbol{R}_{\text{XOR}}$ classifies as $\neg f_1 \vee \neg f_3$. However, $\boldsymbol{R}_{\text{XOR}}$ is different than the one defined in Equation 2.8 because it does not take the combination of instances.



`Figure 4.2:` MILES produced concepts. The chosen instances have a small green diamond over them, the larger green circles indicate the decision boundary of the concept classifiers. Note, though two concepts should have been enough, the MILES classifier still produced five concepts.

The case of the difference between two rule sets comes from the data supplied to the Prism algorithm. In Equation 2.8, there is a hidden extra assumption, namely that bags that contain instances outside these concepts are automatically excluded. Another way to look at it is that Equation 2.8 invalidates the trivial solution of a bag that has neither concept. This could for example be a bag with no instances or a bag with noisy instances outside the defined concepts.

In order to ignore the trivial solution, it's handy to add an additional bag $\boldsymbol{B}_0$ to the bag concept matrix:

$$\boldsymbol{C}_{\boldsymbol{B}} = \begin{bmatrix} f_1(\boldsymbol{B}_1) & \dots & f_{N_C}(\boldsymbol{B}_1) \\ \vdots & \ddots & \vdots \\ f_1(\boldsymbol{B}_{N_{\boldsymbol{B}}}) & \dots & f_{N_C}(\boldsymbol{B}_{N_{\boldsymbol{B}}}) \\ 0 & 0 & 0 \end{bmatrix}, \tag{4.1}$$

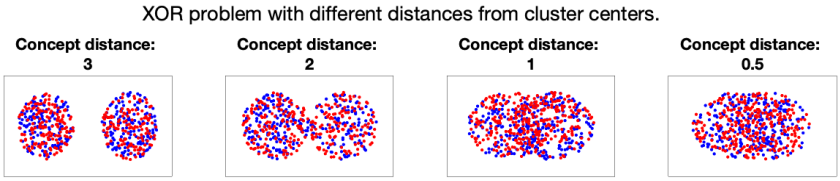
and set the label of $\boldsymbol{B}_0$ to the negative class. The Prism algorithm should now exclude the zero solution because a rule set that excludes the zero solution is more precise. If the Prism algorithm is then run again on the bag concept matrix, the following rule set is generated:

$$\boldsymbol{R}_{\text{XOR}} = \begin{cases} r_1 : f_1 \wedge \neg f_3, \\ r_2 : \neg f_1 \wedge f_3, \end{cases}$$

which is equal to Equation 2.8 if you take into account that f_1 refers to concept A and f_3 refers to concept B.

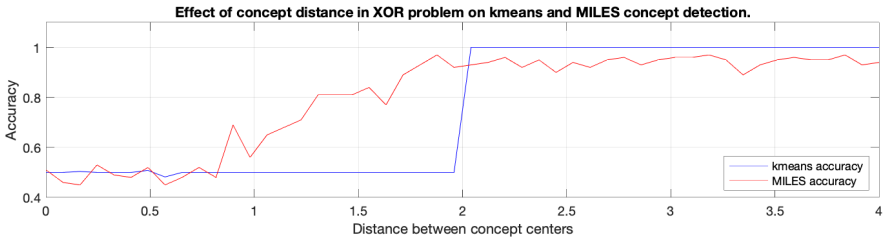
4.2 Concept overlap

By decreasing the distance of concepts in the MIL XOR problem, the difficulty of finding the correct concepts increases. An example is shown in Figure 4.3. Because of the overlapping instances from both classes, it becomes very tough for the concept classifiers to find the actual concepts as soon as the concepts start connecting. It could even be said that the XOR behavior is not present anymore if the concepts completely overlap. Note that the accuracy starts at 0.5 because when the cluster completely overlaps, both CRI methods will default to a single class, which is fifty percent of the data.



`Figure 4.3:` An example of the MIL XOR problem where the distance between the two concepts is reduced. At a lower distance, the concepts start overlapping.

The performance of both concept classifiers in CRI is shown in Figure 4.4. At a distance of 2, the concepts are completely disconnected from each other. MILES outperforms k-means until that point. This is because MILES uses a sparse weighted linear classifier, which can find the concepts before they are totally separated. This is because the distribution of both concepts differs enough at around a distance of 1, which is equal to σ during training. After the concepts are separated, k-means will provide a better split between the two concepts. This allows the rule induction step to perform better.

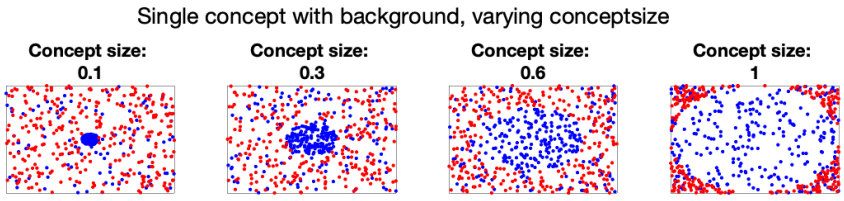


`Figure 4.4:` The performance of the k-means and MILES concept classifiers used in CRI.

4.3 Noisy instances

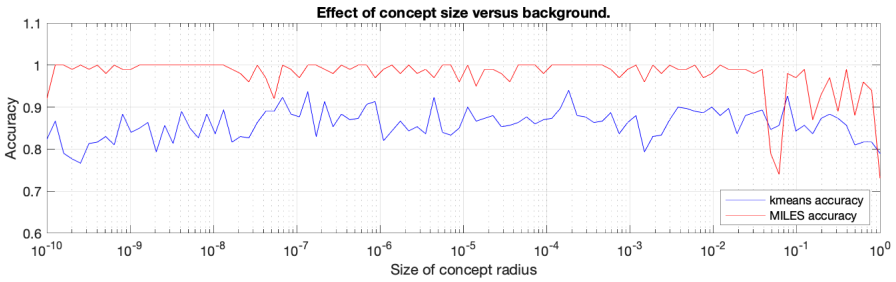
To test how both concept detection methods work in the presence of noise, this toy example will add noise. Noisy instances are instances that don't contribute to the label. In the toy problem, there is a single cluster, which does not contain any noisy instances, meaning that there is a clear separation between the two classes. The toy problem is shown in Figure 4.5. The concept size will be increased, meaning the

surface area of the noise will be reduced. Because the concept here is in direct contact with the noise both methods will perform worse in identifying the concept.



`Figure 4.5:` A noisy one concept MIL problem. A single concept exists in the middle of the graph. The size of this concept is increased for different comparisons for both methods.

The performance is shown in Figure 4.6. It seems that the size of the concept here does not matter that much as the performance of both methods seems reasonably stable. It seems that for both methods the performance drops a bit when the concept reaches size 1. At this point, the noise has shrunken to a very small area. For k-means, this means that the clusters will be put in the corners because there is more data there. And MILES cannot find a good solution to the linear program at this point. Both methods perform better than a random method, which would have an accuracy of 0.5.

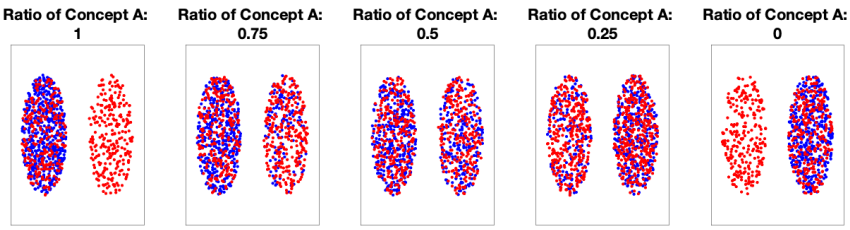


`Figure 4.6:` The performance of CRI on the noisy data set of Figure 4.5, with using k-means or MILES as a concept classifier. MILES outperforms k-means here. This is most likely due to the boundary between the concept and other instances touches, meaning the clustering algorithm fails to find meaningful concepts.

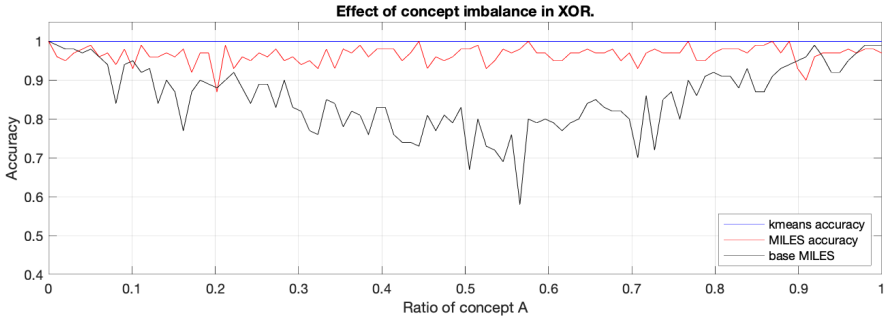
It is too hard for k-means to find a good separation between the cluster and the noise because there are no separable clusters. At higher concept sizes the instances of the negative bags start to form clusters in the corners, but this is still not enough for CRI with k-means to perform well. This means that if there are more noisy instances, MILES will outperform k-means as a concept classifier.

4.4 Concept imbalance

Finally, the ratio of the number of instances in each concept is changed in the XOR problem, as can be seen in Figure 4.7. The performance of CRI can be seen in Figure 4.8. The base MILES algorithm does not perform well for the normal MIL XOR problem, which is expected because a linear combination of instances cannot represent XOR. The performance of all concept detectors is perfect at a ratio of zero and one for concept A. This is because at that point the XOR problem relaxes into the trivial MIL toy problem, explained in Appendix C.



`Figure 4.7:` Concept imbalance XOR data set. In this example toy data set, the amount of bags in a single concept for the XOR class varies. So at a high concept ratio for concept A (left cluster), almost all bags are in concept A, then slowly more bags appear in concept B (right cluster) and fewer in concept A. This data set tests whether a concept classifier can still identify the two concepts. At a ratio of one or two, the data set is similar to the trivial MIL problem, which is shown in Appendix C.



`Figure 4.8:` The performance of Concept rule induction and MILES on a classification problem with concept imbalance as seen in Figure 4.7. For CRI, two different concept classifiers are used. The blue line uses k-means as a concept classifier and the red line uses MILES as a concept classifier. The black line is using base MILES algorithm without any rule induction. The base MILES algorithm does not perform well. CRI using MILES misses a few testing points and k-means has a hundred percent accuracy.

While MILES as a concept classifier in CRI works almost perfectly here, k-means does beat MILES. The concepts are easily separated and while MILES does try to find some concepts it is not perfect. Other forms of MILES are further explored in Appendix D, but from that, it seems that the base MILES still works best in CRI.

4.5 Choosing a concept classifier

MILES performs better than k-means when the concepts cannot be represented in separable clusters or if the data set contains many noisy instances. In the next sections, chapter 5 and chapter 6, k-means will be used as the concept detector. This is because the data in those sections contain separable clusters and the amount of noise is relatively low. If CRI would be used for data that contains overlapping concepts or clusters then it would be better to use MILES instead.

Chapter 5

Rule induction

An advantage of CRI is that the algorithm outputs a list of rules based on concepts. These rules are if-then statements, and it is possible to dive into the "if" part of these statements. For example, when an image number dataset is used, the "if" part of the rules should correspond to identifiable parts of numbers. This section will explore the output of CRI by using two different MIL toy data sets and also show that Prism can produce rules as expected.

This chapter will use the MNIST handwritten digit data set and transform this into two toy MIL data sets [3]. The chapter is then divided into two parts. In the first part, the performance of CRI is examined on the two toy data sets. The second part examines the rules generated for both toy problems. For all experiments, k-means is used as a concept detection tool and Prism for rule induction.

5.1 Concept Rule Induction performance

The standard MNIST handwritten digit dataset will be modified to a MIL dataset [3]. The data set modification will be done in two different ways. The transformations are discussed briefly in this section but the details are given in Appendix C.

The first transformation is based on position. Each image is divided into a seven-by-seven grid, and the average gray-scale value is measured for each four-by-four cell. Cells with a low average gray-scale values are thrown away with the idea that they are part of the dark background and do not hold interesting visual information. The features of each instance are the positional coordinates of the leftover cells. This dataset is called pos-MILMNIST.

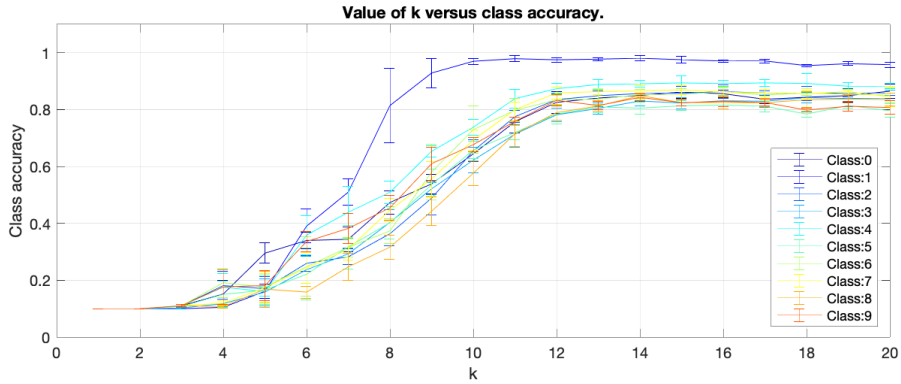
The second transformation is based on random patch sampling. From each image, a set of random eight-by-eight patches is taken. These samples represent parts of the original image, and each instance corresponds to the gray-scale values of each patch. Again, dark patches are thrown away, and the leftover patches make up the instance of the bag. This dataset is called patch-MILMNIST.

Different values of k are tried five times to measure the performance due to the random initialization of k-means. And because Prism works by generating rules in a one-against-all fashion, the performance metric will be class accuracy. For each class, fifty examples are used for the train and test set. The class accuracy for pos-MILMNIST for different values of k is shown in Figure 5.1, and that of patch-MILMNIST is shown in Figure 5.2.

For pos-MILMNIST, identifying class zero against the rest is significantly easier. Around $k = 16$, the testing accuracy stabilizes.

For patch-MILMNIST, the improvement in accuracy plateaus earlier at $k = 12$. Again, class zero seems to outperform the other classes significantly. In both cases, the

`Figure 5.1:` The test class accuracies on the pos-MILMNIST data, using 50 training and testing examples per class.



`Figure 5.2:` The test class accuracies on the patch-MILMNIST data, using fifty training and testing examples per class.

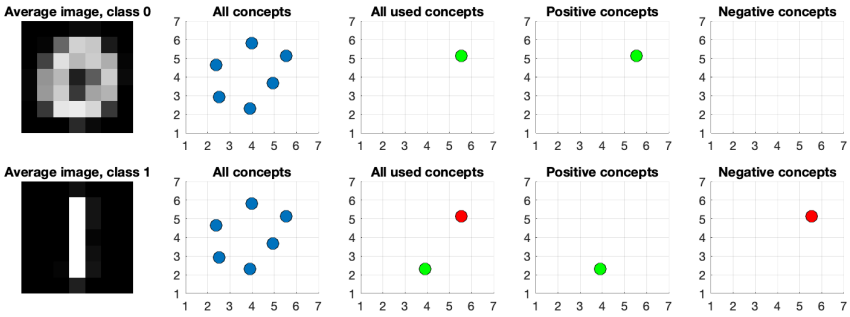
average class accuracy holds at around 0.8. This performance is not as good as recent algorithms for MNIST, which can achieve an accuracy above 0.99 [56]. However, this section aims to see if the rules produced by CRI output conditions correspond to the expected structure of the MNIST dataset so accuracy is not the main focus.

5.2 MILMNIST rule sets examined

The data used in this section will be different than that of the previous sub-section. Each class added to the data set introduces more rules generated by Prism to differentiate between all numbers. To keep the number of rules low, only classes zero, and one will be included in the training and testing set here. To add to that, these classes have apparent visual differences, making them good candidates for examining the IF conditions of these rules.

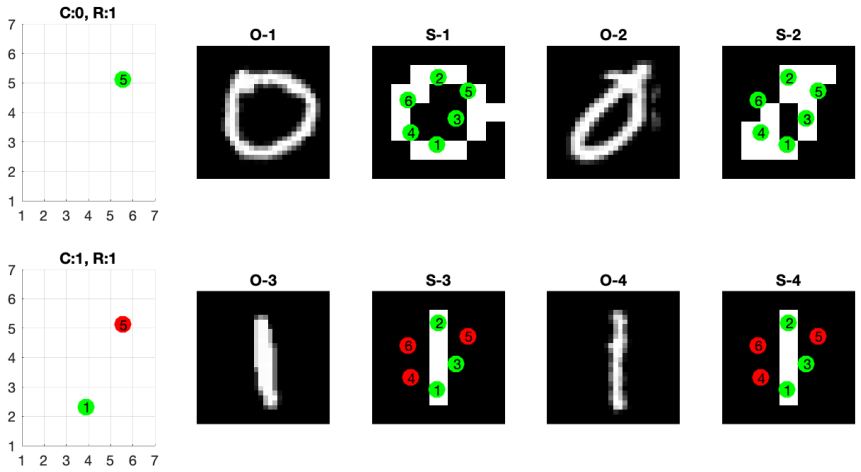
5.2.1 Rule set of pos-MILMNIST

For pos-MILMNIST, the value of k is set to six. In Figure 5.3, it can be seen that this is enough to generate simple rules. Six concepts are found due to the value of k, which lies on the average positional values of the zero class. The rule set of the zero class only needs a positive concept to the right, while the rule set of the one class requires a negative concept.



`Figure 5.3:` The concepts created for class zero and one, including the average MILMNIST sample. The x and y positions are the cluster centers from the sub-sampling. The green and red circles correspond respectively to positive and negative inclusion in the rule sets created by Prism.

It is possible to further zoom in on these rule sets. Each class has a rule set containing a single rule. See Figure 5.4 for the rules and some training examples that apply to the rule. The rule for class one indicates that there must be a white cell at the bottom middle of the original image, but there must not be a white cell simultaneously on the right top of the picture. This rule makes sense because the digit one can be written with a top line from the left before completing the vertical line of the digit one itself.

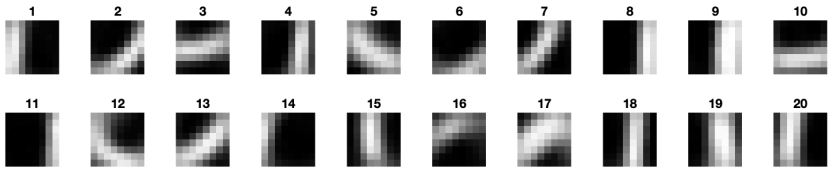


`Figure 5.4:` The rule sets generated for classes one and zero with examples. In this image, C stands for the class, R stands for the rule number, O stands for the original image, and S stands for the MIL sampled image. Each concept has a number and a color. The green color corresponds to the appearance of a concept, and the red color to the absence of it. The concepts used in the rules are shown on the left-most scatter plot. In the S images, the concepts found in the image are shown.

Thus for pos-MILMNIST, the pipeline can find rules corresponding to the class zero and one behavior.

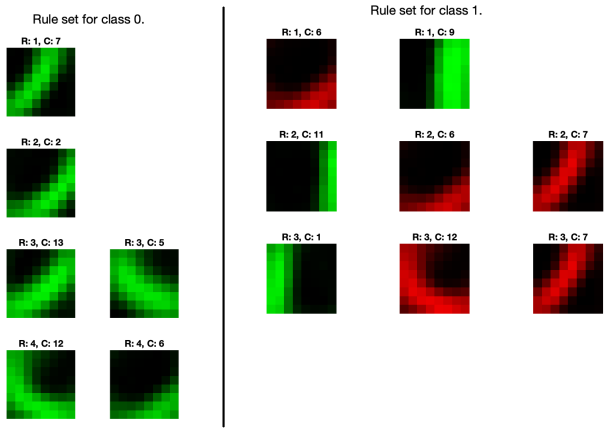
5.2.2 Rule set of patch-MILMNIST

For patch-MILMNIST, the value of $k = 20$ was used. The cluster concepts can be visualized as a small image, which can be seen in Figure 5.5. These cluster centers seem to correspond to vertical lines and circle parts, which makes sense because most *ones* are straight lines in the MNIST data, and a *zero* is a circle.



`Figure 5.5:` The generated concepts from using $k = 20$ with k-means on patch-MILMNIST data.

The rule sets for both classes can be seen in Figure 5.6. For class zero, there are four different rules, each rule only has positive concepts, and there are all arches of circles. The rule set of class one does contain negative concepts. Bags containing any concept that indicates the negative concepts exclude a circle-like part. The positive concepts must appear in a bag; in this case, the positive concepts are all vertical lines.



`Figure 5.6:` The rule set generated by Prism for patch-MILMNIST for class zero and one. The green color indicates that a concept must appear in a bag, and the red color indicates that a concept cannot appear in a bag. Note that *ones* require some vertical concept but without a corner concept.

Prism selects relevant patches which correspond naturally to parts of the two digits.

5.3 Limitations and use of rule analysis

It must be noted that there are some limitations to this approach. When looking at just two MNIST classes, it seems that k-means and Prism can generate rule sets that align with the structure of those digits. The results become harder to analyze when more classes are added. For example, instead of just looking at class zero and one for patch-MILMNIST, the rule set for class zero has 36 rules with, on average, 11 concepts per rule. Other considerations are as follows:

1. These examples can be easily visualized due to their low dimensionality or image representation. Visualization is not always possible. MIL data sets often have high dimensionality. Higher dimensional data makes analyzing the results of the Prism algorithm non-trivial.

2. As seen, each representation of data needs a separate analysis. Comparing rules must be made specifically for a data set.

3. For each training set in this section, only a portion of the total MNIST dataset was used. Adding more training examples results in more variation, needing more rules to separate the classes. Prism makes rules until it covers the entire training set. The additional rules lead to over-fitting, and rule pruning is then required. Several other algorithms focus on rule pruning, such as RIPPER [54], but these bring more hyper-parameters to tune as a disadvantage.

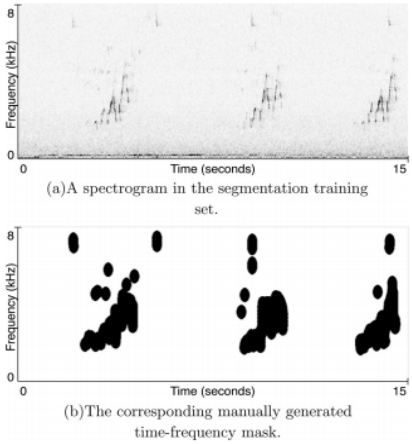
4. The training and testing sets were sampled randomly per class from MNIST. Each random sample can hold digits of slightly different shapes, resulting in other rule sets.

In the following chapter, chapter 6, the rule output of CRI on an actual data set is examined in a similar fashion to the MILMNIST data set of this chapter. The goal will be to show that rules generated on a real-life high-dimensional MIL data set can still be interpreted using k-means and Prism.

Chapter 6

Classification of multiple simultaneous bird species

An example of real data for which CRI could be applicable is the bird-song data set created by Brigs Et al. in [1]. This paper contains sound recordings of bird-song, where multiple birds can sing in the same sound fragment. To transform sound data to a MIL data set, sound fragments of a fixed time interval are exported to a spectrogram, and then a Gaussian filter is used to create *sound blobs*. These blobs can be seen in Figure 6.1.



`Figure 6.1:` The sound data of bird songs, where the sound spectrogram is converted to sound 'blobs' using a gaussian filter, from [1].

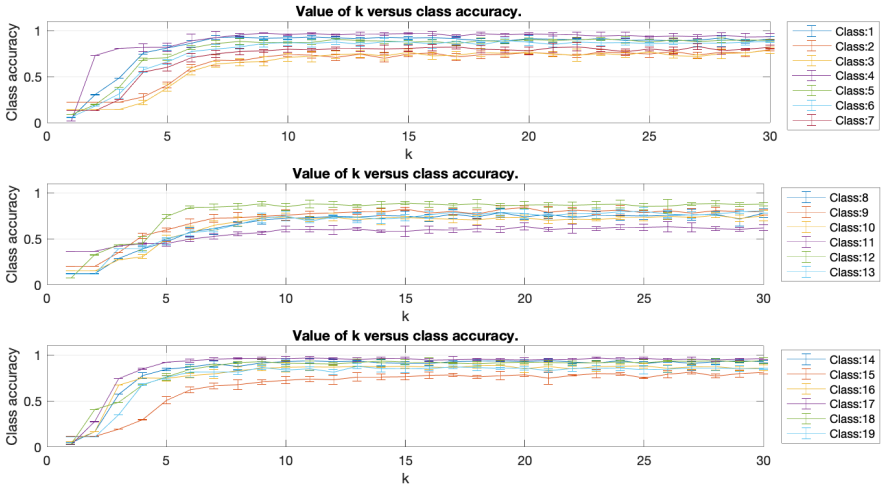
The data used for this paper was retrieved from a Kaggle competition from 2013 [2]. This dataset contained the bounding boxes of the blobs from Figure 6.1, which will be helpful while examining the rule sets from CRI. The blobs are put through the feature extraction pipeline described in the original paper by Brigs Et al. [1]. This transformation creates a multi-instance multi-label data set, where each sound blob represents an instance, and each bag is a sound fragment of ten seconds.

Again, the goal of this chapter, much like chapter 5, is to examine the rules generated by CRI. This dataset is a good candidate because these sound blobs CRI creates can be traced back to the sounds produced by birds. An expert could listen to these smaller sound fragments in the IF conditions of rules and thus check the generated rules. Unfortunately, the author of this paper is no bird song expert, so instead, this paper will examine the rules created on the spectrogram.

The next section will first look at the performance of the data set by looking at the class accuracies. In the second section, a few examples are shown using the spectrogram. One of the curiosities that occurred during testing is that the clustering parameter k significantly influences how rules are produced. Thus in the third section, the impact of the clustering parameter k is examined.

6.1 CRI class performance on bird song

The MIL data set is put through CRI as described in Figure 3.1. For the concept classifier, k-means will be used, and this requires the instance features to be scaled per feature between 0 and 1. For the rule induction, Prism will be used. Similarly to section 5.1, average class accuracy will be used to evaluate the performance of k. Each experiment will be run five times to account for the effect of different k-means initialization.



`Figure 6.2:` Test class performance of all 19 bird classes from the Kaggle data set using the competition train/test folds [2]. Because there are 19 classes, the plot is divided into subplots.

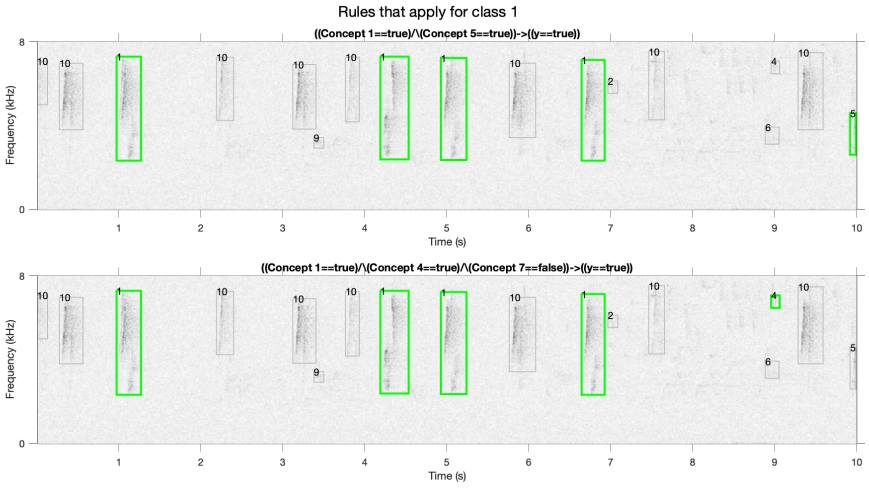
The class performance is shown in Figure 6.2. Firstly, at $k = 1$, different class performances are measured. The performance difference results from all class frequencies not being the same. More prevalent classes will have a higher base class accuracy because they appear more often. It seems that a performance plateau is reached when $k = 10$. After this point, the testing accuracy is stable even when k approaches the number of train instances. Notably, the performance of class 11 is significantly lower than the rest, most likely because class 11 rarely appears alone.

The performance of CRI on the bird data performance is not as good as that of the original paper [1]. CRI achieves an average class accuracy of 0.87, while the MILkNN method managed an average class accuracy of 0.90. While the accuracy is slightly lower for CRI, CRI does create rule sets. The following sections will look at the rules that are produced.

6.2 Rule visualization

Rule sets on real-life data have to be visualized differently than the MILMNIST data sets. One way of doing this is to keep track of the position of instances in the original spectrograms. The k-means concept classifier will assign one of the k-means cluster centers to the closest instance for each bag. Because Prism uses these concept presences to create rules, it is possible to see what rules apply to each bag.

A way to visualize this can be seen in Figure 6.3. Two rules from the class rule set apply to this bag. The concept number is shown on the top left of each bag, and the original instance sound blob is within the bounding box. Sound instances that are required by a rule are marked green.



`Figure 6.3:` The rules that apply for class one on a certain bag, which contains that class shown on bounding boxes of instances on the original spectrogram of 10 seconds of bird song. The rules are sounds that belong together for a specific bird species. Original data from [2].

It seems that for class 1, for other instances (not shown here), it is not enough to have just concept 1. It seems that concept 5 and concept 4 also appear together with class 1 in the training set. Also, note that rule 2 requires an additional restriction, where concept 7 is not allowed.

A possible application of these rules would be to extract the sound fragments for a specific class. Finding the sound fragments can be done by looking at the positive concepts in the rules in each class. These sound fragments could then make up the song of a bird. However, a value of $k = 10$ is too low as many different negative concepts are created here. We need to look at higher k values for this approach to work.

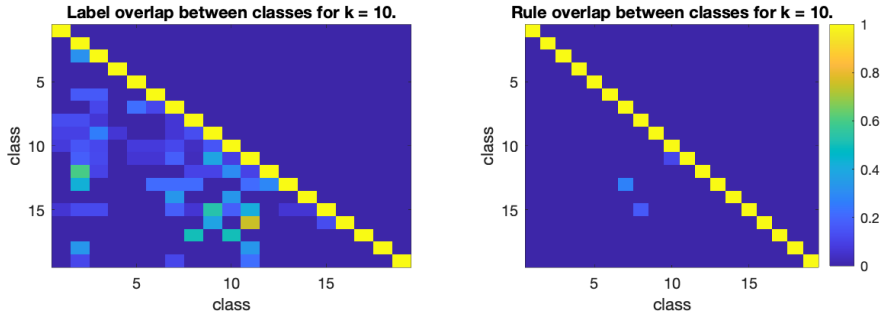
6.3 Influence of the clustering parameter

As seen in the previous subsection, the cluster parameter k and the resulting class accuracy stabilize for $k > 10$, even though one might suspect that the 19 birds produce more than ten different sounds. So why can the CRI algorithm still get decent performance for just $k = 10$? This subsection will first check if class co-occurrence is an issue. Class co-occurrence can result in CRI with only a few rules because it only needs to recognize the co-occurrence. The second issue that will be examined is

the appearance of negative concepts because it is unexpected to identify a bird song without sounds.

Starting with class co-occurrence, if several birds are only found in the same area, then it would make sense they co-occur in the same sound fragments. In this case, for birds with high co-occurrence, the same rules could be used to identify them. To examine this, the class co-occurrence in the multi-label data set is here with the rule overlap between class rule sets in Figure 6.4.

In the bird data set, there seem to be some bird classes that highly overlap, most prominently classes 11 and 16. An example of a class usually appears alone is class 4.



`Figure 6.4:` On the left: class co-occurrence of the bird data set. For each class in the bird data set, the number of times another class occurs in the same sound fragment, bag, is counted and then divided by the number of times the original class occurs. On the right: similar, comparing the rule sets produced by CRI and counting how many times rules overlap between the class rule sets. It shows that there is no correlation between co-occurrence and rule overlap

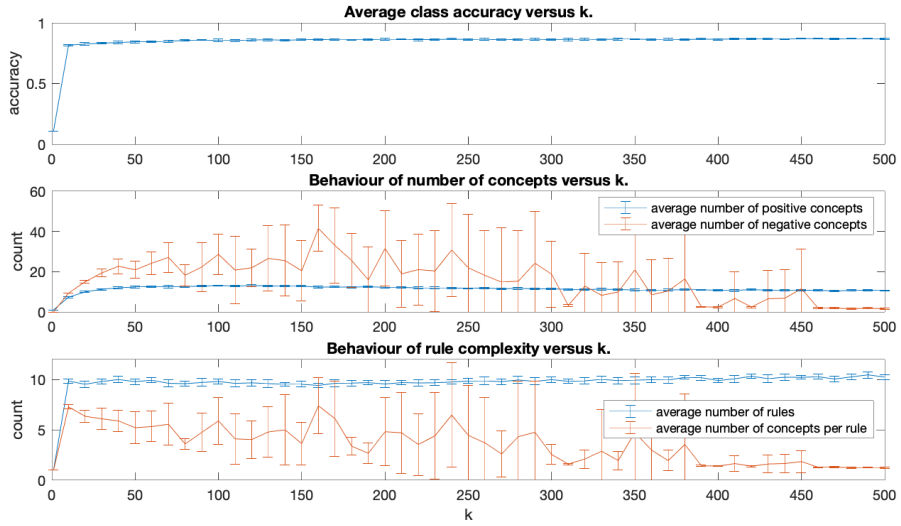
From Figure 6.4, there is no clear correlation between the overlap of rules and class co-occurrence. The figure on the right is almost zero, meaning that there are nearly no overlapping rules produced by CRI, even for the most co-occurring classes, 11 and 16. Thus the performance of CRI at $k = 10$ is not explained by class co-occurrence.

The second part of CRI that can be checked is the negative concepts used in the rules that CRI generates. Classifying birds with negative concepts would mean that the classifier uses the absence of certain bird sounds to identify other birds. Classification by absence can only happen if the presence of certain birds negates the presence of other birds.

The number of clusters of k can also be further increased to examine how it affects the number of negative concepts used in the rulesets. The average of all classes is taken to keep the plot readable. This plot can be seen in Figure 6.5. For k, steps of 10 are chosen to give a clear line. Also, note that the test performance keeps improving slightly and is the highest on $k = 500$.

The average number of positive concepts in a class rule set stabilizes around 10 for $k > 100$. But the average number of negative concepts rises initially, then decreases at $k > 450$ to about one. There are 1003 instances in the training set, so roughly two instances per cluster is a small cluster. At that point, CRI is looking specifically at the individual instances. At $k = 500$, almost no negative instances are used anymore, which might indicate that, at this point, concepts found are more likely to be actual sound fragments produced by the birds of that class. The number of concepts corresponds to the number of different bird sounds in the original competition [2].

Furthermore, in Figure 6.5, the rule complexity and the number of rules is also plotted against k. While the number of rules needed stays roughly the same for each



`Figure 6.5:` On the top, the class test accuracy for different values of k used in k-means CRI. In the middle, the average number of positive and negative concepts used in the class rule sets for different values of kl. And on the bottom, the average number of rules per class and the average number of concepts for rules per class for different values of k. It seems that for $k > 450$, the rules become made with fewer and fewer negative concepts.

class, the number of concepts used goes down to around one. Since, at this point, most concepts are positive, this means that rules now identify a single sound fragment of that bird.

At lower k values, the CRI algorithm needs to use the absence of certain sounds to classify classes correctly, but it does manage to have still a performance of an average class accuracy of 0.87. At higher k values, the performance does not drop, so CRI starts to identify instances (sound blobs) to belong to certain classes instead. All in all, it seems that CRI can work for the bird dataset, but the meaning of the rules changes for different values of k. Using CRI for a real-life data set, it is crucial to try and investigate the rules produced by the algorithm to see if the rules make sense.

To conclude this chapter, the performance of CRI on the real-life bird song MIL data set does not beat the original paper [1], but the rules generated allow us to trace back classification results to sets of bird song fragments. The influence of the clustering parameter k has high importance on the validity of the rules. If k is put too low, the rule sets for a specific bird species will contain many negative (absent) concepts, which does not make sense when trying to find the sound fragments of a particular bird species song. At higher values of k, it is shown that these negative concepts are no longer needed for Prism to cover classes and that actual combinations of the sound fragment of bird song do occur. And lastly, it was shown that these sound fragments have almost no overlap between bird species.

Chapter 7

Conclusion and discussion

Multiple Instance Learning data (MIL) sets only contain information on groups (bags) of data points (instances). In MIL, This paper proposes a new MIL classifier called Concept Rule Induction (CRI). The main research question of this paper is whether CRI could create a MIL classifier with improved interpretability using concepts. The idea behind a *concept* is that concepts are properties of a specific class that is shared between bags of that class, and not shared between bags of other classes. CRI is a two-step algorithm, that first detects concepts, and then performs rule induction on those concepts. MILES and k-means were used as concept detection algorithms and Prism was used as a rule induction algorithm. CRI was examined on several toy data sets and a real-life data set on tuning, performance, and interpretability. This chapter will go over the findings on CRI and discuss improvement points of CRI.

CRI was tested on the MIL eXclusive OR (XOR) toy problem, which is difficult for classical MIL classifiers. This problem pitted the two concept classifiers, MILES and k-means, against each other. The MIL XOR toy problem showed how to tune the concept classifiers. CRI is able to solve MIL XOR perfectly, with MILES performing better on noisy and overlapping instances and k-means performing better otherwise.

Next, the rule induction of CRI was tested on two MIL MNIST number toy classification problems and a real-life multi-label MIL bird song classification problem. For all three problems, CRI is able to find interpretable rules. For the MIL MNIST number problems, CRI finds concepts that represent circles and lines that create numbers and then combines them in a logical fashion to the final number classification. For the MIL bird song data, CRI finds individual sound fragments which make up the song of a specific bird. The performance of CRI is currently lower than state-of-the-art algorithms, for the bird song data CRI managed an average class accuracy of 0.87 versus the 0.90 average class accuracy of the MILkNN algorithm used in the original bird song paper. The performance reduction is a result of the use of hard cut-off points in the concept detection algorithms. In conclusion, CRI does add improved interpretability but at the cost of an accuracy loss.

To improve CRI, one could first look at the few tuneable parameters of the MILES concept classifier. The MILES concept classifier did outperform the k-means classifier in the presence of noise or class imbalance but was harder to fine-tune and k-means did outperform miles on the MIL MNIST and bird song data. The optimization constraints of the linear sparse classifier used by MILES can take class imbalance into account. This could be useful in the bird song data set because some birds only appear less than ten times, and others over a hundred times. By using the extended optimization constraints that do take class imbalance into account the performance of MILES should improve.

Another parameter of MILES is the scaling factor σ, which significantly influences how the MILES concept classifier is defined. The scaling factor is used both in the definition of similarity and in the concept classification. In the Appendix D, the following two schemes were tried: setting the scaling factor per instance using instances in its neighborhood and using a binary similarity method with σ instead. These two approaches did not seem to give better performance overall. If the similarity metric could be improved, it would improve the performance of CRI when using MILES as the concept classifier and MILES itself.

And as mentioned, one could look at other concept-generating algorithms. Diverse Density would be an option, but it is computationally expensive. k-Means could be adapted by only running it on the positive bags of a specific class, this would mean that clustering must be done per class now, but it does more with the definition of a concept. If there is a lot of noise, the clustering algorithm HDBSCAN specifically adds noise protection as outliers instances are not clustered [57]. Using one of these algorithms as a concept detector would reduce the number of rules CRI makes, but the impact on performance must be studied.

The rule induction algorithm could be improved as Prism is just the baseline rule inductor algorithm. There are more advanced rule induction algorithms such as FURIA, that induce fuzzy rules [58]. Fuzzy rules would reduce interpretability, but fuzzy rules could deal with concept overlap and noise. The amount of training data can also be low, which might result in a weak rule set. These issues could be tackled by making rules more general by changing the rule selection criterion. RIPPER would be an example of this [59].

To further extend the MIL assumption, it might be possible to look at temporal logic. For example, sound fragments have an order in the bird-song MIL data set, and a low-pitch sound might always be followed by a high-pitch sound and then some silence for a specific bird species. In this case, it might be helpful to create temporal logic in the rule sets to account for these kinds of sound patterns over time [60, 61].

Bibliography

[1] F. Briggs, B. Lakshminarayanan, L. Neal, X. Z. Fern, R. Raich, S. J. Hadley, A. S. Hadley, and M. G. Betts, "Acoustic classification of multiple simultaneous bird species: A multi-instance multi-label approach," *The Journal of the Acoustical Society of America*, vol. 131, no. 6, pp. 4640–4650, 2012.

[2] Y. Huang, F. Briggs, R. Raich, K. Eftaxias, and Z. Lei, "The ninth annual mlsp data competition," in *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2013, pp. 1–4.

[3] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[4] M.-A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon, "Multiple instance learning: A survey of problem characteristics and applications," *Pattern Recognition*, vol. 77, pp. 329–353, 2018.

[5] B. Babenko, "Multiple instance learning: algorithms and applications," *View Article PubMed/NCBI Google Scholar*, pp. 1–19, 2008.

[6] P. Chikontwe, M. Kim, S. J. Nam, H. Go, and S. H. Park, "Multiple instance learning with center embeddings for histopathology classification," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2020, pp. 519–528.

[7] P. Sudharshan, C. Petitjean, F. Spanhol, L. E. Oliveira, L. Heutte, and P. Honeine, "Multiple instance learning for histopathological breast cancer image classification," *Expert Systems with Applications*, vol. 117, pp. 103–111, 2019.

[8] M. Lerousseau, M. Vakalopoulou, M. Classe, J. Adam, E. Battistella, A. Carré, T. Estienne, T. Henry, E. Deutsch, and N. Paragios, "Weakly supervised multiple instance learning histopathological tumor segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2020, pp. 470–479.

[9] X. Wang, F. Tang, H. Chen, L. Luo, Z. Tang, A.-R. Ran, C. Y. Cheung, and P.-A. Heng, "Ud-mil: uncertainty-driven deep multiple instance learning for oct image classification," *IEEE journal of biomedical and health informatics*, vol. 24, no. 12, pp. 3431–3442, 2020.

[10] J. Yao, X. Zhu, J. Jonnagaddala, N. Hawkins, and J. Huang, "Whole slide images based cancer survival prediction using attention guided deep multiple instance learning networks," *Medical Image Analysis*, vol. 65, p. 101789, 2020.

[11] M. Ilse, J. M. Tomczak, and M. Welling, "Deep multiple instance learning for digital histopathology," in *Handbook of Medical Image Computing and Computer Assisted Intervention*. Elsevier, 2020, pp. 521–546.

[12] Z. Han, B. Wei, Y. Hong, T. Li, J. Cong, X. Zhu, H. Wei, and W. Zhang, "Accurate screening of covid-19 using attention-based deep 3d multiple instance learning," *IEEE transactions on medical imaging*, vol. 39, no. 8, pp. 2584–2594, 2020.

[13] Z. Li, W. Zhao, F. Shi, L. Qi, X. Xie, Y. Wei, Z. Ding, Y. Gao, S. Wu, J. Liu *et al.*, "A novel multiple instance learning framework for covid-19 severity assessment via data augmentation and self-supervised learning," *Medical Image Analysis*, vol. 69, p. 101978, 2021.

[14] W. Xue, C. Cao, J. Liu, Y. Duan, H. Cao, J. Wang, X. Tao, Z. Chen, M. Wu, J. Zhang *et al.*, "Modality alignment contrastive learning for severity assessment of covid-19 from lung ultrasound and clinical information," *Medical image analysis*, vol. 69, p. 101975, 2021.

[15] Y. Wang, J. Li, and F. Metze, "A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 31–35.

[16] S. Mao, P. Ching, and T. Lee, "Deep learning of segment-level feature representation with multiple instance learning for utterance-level speech emotion recognition." in *Interspeech*, 2019, pp. 1686–1690.

[17] M. Aktas, A. Bayramcavus, and T. Akgun, "Multiple instance learning for cnn based fire detection and localization," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2019, pp. 1–8.

[18] X. Li, Y. Lang, Y. Chen, X. Mao, Y. He, S. Wang, H. Xue, and Q. Lu, "Sharp multiple instance learning for deepfake video detection," in *Proceedings of the 28th ACM international conference on multimedia*, 2020, pp. 1864–1872.

[19] T. Zhao, X. Xu, M. Xu, H. Ding, Y. Xiong, and W. Xia, "Learning self-consistency for deepfake detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 15 023–15 033.

[20] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018.

[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[22] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[24] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[25] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 3387–3395.

[26] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[27] Y. Fenjiro and H. Benbrahim, "Deep reinforcement learning overview of the state of the art." *Journal of Automation, Mobile Robotics and Intelligent Systems*, pp. 20–39, 2018.

[28] Y. Chevaleyre and J.-D. Zucker, "Noise-tolerant rule induction from multi-instance data," in *ICML 2000, workshop on attribute-value and relational learning*. Citeseer, 2000.

[29] A. Srinivasan, S. H. Muggleton, M. J. Sternberg, and R. D. King, "Theories for mutagenicity: A study in first-order and feature-based induction," *Artificial Intelligence*, vol. 85, no. 1-2, pp. 277–299, 1996.

[30] Z.-H. Zhou, "Multi-instance learning: A survey," *Department of Computer Science & Technology, Nanjing University, Tech. Rep*, vol. 1, 2004.

[31] J. Foulds and E. Frank, "A review of multi-instance learning assumptions," *The Knowledge Engineering Review*, vol. 25, no. 1, pp. 1–25, 2010.

[32] Y. Chen, J. Bi, and J. Z. Wang, "Miles: Multiple-instance learning via embedded instance selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 1931–1947, 2006.

[33] H. K. Büning and T. Lettmann, *Propositional logic: deduction and algorithms*. Cambridge University Press, 1999, vol. 48.

[34] J. N. Washburne, "The definition of learning." *Journal of Educational Psychology*, vol. 27, no. 8, p. 603, 1936.

[35] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," *Machine learning*, pp. 3–23, 1983.

[36] X. Chen and T. Han, "Disruptive technology forecasting based on gartner hype cycle," in *2019 IEEE Technology & Engineering Management Conference (TEMSCON)*. IEEE, 2019, pp. 1–6.

[37] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

[38] A. Sorokin and D. Forsyth, "Utility data annotation with amazon mechanical turk," in *2008 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE, 2008, pp. 1–8.

[39] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk," in *Proceedings of the ACM SIGKDD workshop on human computation*, 2010, pp. 64–67.

[40] H. Steinhaus, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci*, vol. 1, no. 804, p. 801, 1956.

[41] G. E. Hinton, T. J. Sejnowski *et al.*, *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[42] X. J. Zhu, "Semi-supervised learning literature survey," *University of Wisconsin-Madison Department of Computer Sciences*, 2005.

[43] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," *Advances in neural information processing systems*, pp. 570–576, 1998.

[44] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial intelligence*, vol. 89, no. 1-2, pp. 31–71, 1997.

[45] M.-A. Carbonneau, E. Granger, A. J. Raymond, and G. Gagnon, "Robust multiple-instance learning ensembles using random subspace instance selection," *Pattern recognition*, vol. 58, pp. 83–99, 2016.

[46] Y. Xiao, B. Liu, and Z. Hao, "A sphere-description-based approach for multiple-instance learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 2, pp. 242–257, 2016.

[47] N. Weidmann, E. Frank, and B. Pfahringer, "A two-level learning method for generalized multi-instance problems," in *European Conference on Machine Learning*. Springer, 2003, pp. 468–479.

[48] J. Mingers, "Expert systems—rule induction with statistical data," *Journal of the operational research society*, vol. 38, no. 1, pp. 39–47, 1987.

[49] O. Maron, "Learning from ambiguity," Ph.D. dissertation, Massachusetts Institute of Technology, 1998.

[50] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, vol. 20, 2012.

[51] T. S. Madhulatha, "An overview on clustering methods," *arXiv preprint arXiv:1205.1117*, 2012.

[52] C. Molnar, *Interpretable Machine Learning*. Lulu. com, 2020.

[53] J. Cendrowska, "Prism: An algorithm for inducing modular rules," *International Journal of Man-Machine Studies*, vol. 27, no. 4, pp. 349–370, 1987.

[54] W. W. Cohen, "Fast effective rule induction," in *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[55] J. R. Quinlan, "Discovering rules by induction from large collections of examples," *Expert systems in the micro electronics age*, 1979.

[56] A. Baldominos, Y. Saez, and P. Isasi, "A survey of handwritten character recognition with mnist and emnist," *Applied Sciences*, vol. 9, no. 15, p. 3169, 2019.

[57] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *Journal of Open Source Software*, vol. 2, no. 11, p. 205, 2017.

[58] J. Hühn and E. Hüllermeier, "Furia: an algorithm for unordered fuzzy rule induction," *Data Mining and Knowledge Discovery*, vol. 19, no. 3, pp. 293–319, 2009.

[59] M. Sasaki and K. Kita, "Rule-based text categorization using hierarchical categories," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, vol. 3. IEEE, 1998, pp. 2827–2830.

[60] A. Brunello, G. Sciavicco, and I. E. Stan, "Interval temporal logic decision tree learning," in *European Conference on Logics in Artificial Intelligence*. Springer, 2019, pp. 778–793.

[61] D. Bresolin, E. Cominato, S. Gnani, E. Muñoz-Velasco, and G. Sciavicco, "Extracting interval temporal logic rules: A first approach," in *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[62] Y. Chevaleyre and J.-D. Zucker, "Solving multiple-instance and multiple-part learning problems with decision trees and rule sets. application to the mutagenesis problem," in *Conference of the Canadian Society for Computational Studies of Intelligence*. Springer, 2001, pp. 204–214.

[63] J. Novovicová, P. Pudil, and J. Kittler, "Divergence based feature selection for multimodal class densities," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 18, no. 2, pp. 218–223, 1996.

[64] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[65] A. Jain and D. Zongker, "Feature selection: Evaluation, application, and small sample performance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 2, pp. 153–158, 1997.

[66] N. Kwak and C.-H. Choi, "Input feature selection by mutual information based on parzen window," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 24, no. 12, pp. 1667–1671, 2002.

[67] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 301–312, 2002.

[68] M. Bressan and J. Vitria, "On the selection and classification of independent features," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 10, pp. 1312–1317, 2003.

[69] F. J. Iannarilli Jr and P. A. Rubin, "Feature selection for multiclass discrimination via mixed-integer linear programming," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 25, no. 6, pp. 779–783, 2003.

[70] B. Krishnapuram, A. Harternink, L. Carin, and M. A. Figueiredo, "A bayesian approach to joint feature selection and classifier design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1105–1111, 2004.

[71] M. H. Law, M. A. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[72] P. Somol, P. Pudil, and J. Kittler, "Fast branch & bound algorithms for optimal feature selection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 7, pp. 900–912, 2004.

[73] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[74] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[75] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[76] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[77] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[78] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[79] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM review*, vol. 43, no. 1, pp. 129–159, 2001.

[80] J. Bi, K. Bennett, M. Embrechts, C. Breneman, and M. Song, "Dimensionality reduction via sparse support vector machines," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1229–1243, 2003.

[81] A. Smola, B. Scholkopf, and G. Ratsch, "Linear programs for automatic accuracy control in regression," in *9th International Conference on Artificial Neural Networks: ICANN*. IET, 1999, pp. 575—580.

[82] J. Zhu, S. Rosset, R. Tibshirani, and T. J. Hastie, "1-norm support vector machines," in *Advances in neural information processing systems*, 2004, pp. 49–56.

[83] J. Durkin, "Expert systems: a view of the field," *IEEE Intelligent Systems*, vol. 11, no. 02, pp. 56–63, 1996.

[84] J. Liu, S. Ji, and J. Ye, *SLEP: Sparse Learning with Efficient Projections*, Arizona State University, 2009. [Online]. Available: http://www.public.asu.edu/~jye02/Software/SLEP

Appendix A

MILES

In 2006 Y. Chen et all. devised an algorithm that defined the concepts as Gaussian blobs around the training instances [32]. The algorithm used a support vector machine, where the weights indicated which of the Gaussian blobs contribute to the label of the bag. These Gaussian blobs represent possible concepts $c \in \mathcal{C}$. This algorithm was called Multiple Instance Learning via Embedded Instance Selection (MILES). Part of MILES will be used as a concept detector for the proposed MIL classifier in this paper. This appendix will explain the history and inner workings of MILES.

MILES is inspired by the Diverse Density (DD) framework, which was first derived in [43, 49]. DD tries to find a point $\boldsymbol{x} \in \mathbb{R}^d$ to act as a concept, with d being the number of features. Thus the concepts can be any point in feature space $\boldsymbol{x} = c \in \mathcal{C} = \mathbb{R}^d$. If one assumes this concept is a single point it is possible to use probability theory to find it by maximizing $P(x = c|\boldsymbol{B}_1, \ldots, \boldsymbol{B}_N)$. Finding this concept c was done using gradient decent in [49] and with the use of expectation maximization in [62]. However, both these approaches do not guarantee global optimality and thus can get stuck in local maxima. Therefore, many runs with different starting searching points are used, making the algorithms using the DD framework time-consuming.

Y. Chen et all. interpreted this DD framework from a feature subset selection viewpoint. Methods on feature subset selection are well studied in areas of machine learning [63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74]. These methods can be divided into the following two categories: filter and wrapper based feature selection [64, 73, 74]. Filter-based methods are founded in information, similarity, or statistical theory [74]. These methods usually do not guarantee a good classifier performance and can select features that are intercorrelated [32].

Wrappers, on the other hand, use the required classifier in the performance measure of the feature selection. This results in better generalization, but with twice the computational cost [32, 64]. To combat the increased computational cost, MILES uses a 1-norm Support Vector Machine (SVM) to do feature selection and classification at the same time. This 1-norm SVM is formulated as Linear Program (LP). Many free and commercial solvers exist for efficient solving of these linear programs [50]. In order to use a wrapper-based algorithm for MIL problems, the authors of MILES made use of an instance-based feature mapping.

A.1 Instance-based feature mapping

While the DD framework tries to find a single concept, MILES allows for multiple concepts to be identified. MILES does this by treating each instance as a potential concept, with the shape of a Gaussian blob around the instance. Before the concept

identification step, MILES first uses an embedding on the training bags to measure how potential concepts relate to the bags. To start this embedding step, index all n instances of all N training bags as $\boldsymbol{x}^k$, where $l = 1, \ldots, n$. Because each instance $\boldsymbol{x}^l$ is a potential concept, $\mathcal{C}$ is now countable and limited to:

$$\mathcal{C} = \{\boldsymbol{x}^l : l = 1, \ldots, n\}. \tag{A.1}$$

Next, MILES tries to relate the potential concepts $\boldsymbol{x}^l$ to bags as $\mathrm{P}(\boldsymbol{x}^l|\boldsymbol{B}_i)$. In order to estimate $\mathrm{P}(\boldsymbol{x}^l|\boldsymbol{B}_i)$ the authors of MILES used the *most-likely-cause estimator* from the diverse density framework [49]. This estimator is defined as:

$$\mathrm{P}(\boldsymbol{x}^l|\boldsymbol{B}_i) \propto s(\boldsymbol{x}^l, \boldsymbol{B}_i) = \max_j \exp\left(-\frac{||\boldsymbol{x}_{ij} - \boldsymbol{x}^l||^2}{\sigma^2}\right), \tag{A.2}$$

where σ is the scaling factor that depends on the distribution of the data. Each bag $\boldsymbol{B}_i$ in the training data set is embedded into the concept space $\mathbb{F}_{\mathcal{C}}$ as

$$\boldsymbol{m}(\boldsymbol{B}_i) = \left[s(\boldsymbol{x}^1, \boldsymbol{B}_i), s(\boldsymbol{x}^2, \boldsymbol{B}_i), \ldots, s(\boldsymbol{x}^n, \boldsymbol{B}_i)\right]. \tag{A.3}$$

Then, applying this embedding to the training set of n instances and N bags the following matrix is created in concept space $\mathbb{F}_{\mathcal{C}}$:

$$\begin{aligned}
&[\boldsymbol{m}_1, \ldots, \boldsymbol{m}_N] \\
&= [\boldsymbol{m}(\boldsymbol{B}_1), \ldots, \boldsymbol{m}(\boldsymbol{B}_N)] \\
&= \begin{bmatrix} s(\boldsymbol{x}^1, \boldsymbol{B}_1) & \ldots & s(\boldsymbol{x}^1, \boldsymbol{B}_N) \\ \vdots & \ddots & \vdots \\ s(\boldsymbol{x}^n, \boldsymbol{B}_1) & \ldots & s(\boldsymbol{x}^n, \boldsymbol{B}_N) \end{bmatrix},
\end{aligned}$$

so for a single row: $s(\boldsymbol{x}^l, \cdot) = [s(\boldsymbol{x}^l, \boldsymbol{B}_1), \ldots, s(\boldsymbol{x}^l, \boldsymbol{B}_N)]$ one obtains the embedding of instance $\boldsymbol{x}^k$ for all bags.

The idea behind this embedding is that if an instance $\boldsymbol{x}^k$ has high similarity to bags of some class and low similarity to some bags of other classes, then the information introduced by the $s(\boldsymbol{x}^k, \cdot)$ is useful in separation and labeling between the classes. The next step is applying an SVM to do the classification.

A.2 Sparse Support Vector Machine

While CRI should be able to detect logic between concepts and class prediction with multiple labels, this section will start with the two-class problem. This method can be extrapolated to the multi-class setup, with Q classes, by using a so-called one-versus-all combination scheme [75], which fits with CRI creating rulesets per class. This section will construct a linear program that can learn the weights of the sparse linear classifier.

Let's start with a linear classifier:

$$y = \mathrm{sign}\left(\boldsymbol{w}^T\boldsymbol{m} + b\right). \tag{A.4}$$

Here y denotes the class label, which can be either $+1$ or -1 corresponding to the bag label. The input is $\boldsymbol{m}$, which is an embedded bag in concept space $\mathbb{F}_{\mathcal{C}}$ as discussed earlier. Lastly, $\boldsymbol{w}$ and b are the weight vector and the offset.

A Support Vector Machine classifier is a linear classifier that optimizes $\boldsymbol{w}$ and b by minimizing:

$$\lambda P[\boldsymbol{w}] + \mathcal{E}_{training}, \tag{A.5}$$

where λ is the regularization parameter and $\mathcal{E}_{training}$ is defined as the hinge loss:

$$\mathcal{E}_{training} = \xi = \max\{1 - y(\boldsymbol{w}^T \boldsymbol{m} + b), 0\}, \tag{A.6}$$

and $P[\boldsymbol{w}]$ is a regularization term:

$$P[\boldsymbol{w}] = \boldsymbol{w}^T \boldsymbol{w}. \tag{A.7}$$

Regularization is used to keep the weights $\boldsymbol{w}$ small, as huge weights often lead to overfitting on the training data [76, 77]. To add to that, if weights become zero, they can be ignored in the concept detection step of CRI.

The goal of the sparse linear classifier is that points in feature space $\mathbb{F}_{\mathcal{C}}$ which contribute to the labeling of $\boldsymbol{m}$ will have non-zero weights in the optimal solution. And the points in $\mathbb{F}_{\mathcal{C}}$ that do not contribute to the label will have zero weights. This allows us to identify concepts from instances $\boldsymbol{x}^l$. In standard SVMs, the squared 2-norm of the weight vector $||\boldsymbol{w}||_2^2$ is used as the regularization term as defined above. MILES, however, uses the 1-norm of the weight vector instead, i.e.:

$$||\boldsymbol{w}||_1 = \sum_l |w_l|. \tag{A.8}$$

The 1-norm formulation forces more weights to zero than the 2-norm regularizer. As an added bonus, the 1-norm formulation is also linear instead of quadratic, making it faster. This idea is also used in other techniques such as LASSO [78], basis pursuit [79] and drug discovery [80].

If there are ℓ^+ positive samples and ℓ^- negative samples in the training data set and let $\boldsymbol{m}_i^+$ be an example of an embedding of a positive bag $\boldsymbol{B}_i^+$, similarly $\boldsymbol{m}_j^-$ for a negative bag $\boldsymbol{B}_j^-$. Then the SVM training problem can be formulated as:

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi,\eta} \quad & \lambda \sum_{k=1}^{n} |w_k| + C_1 \sum_{i=1}^{\ell^+} \xi_i + C_2 \sum_{j=1}^{\ell^-} \eta_j \\ \text{subject to} \quad & (\boldsymbol{w}^T \boldsymbol{m}_i^+ + b) + \xi_i \geq 1, \quad i = 1, \ldots, \ell^+, \\ & -(\boldsymbol{w}^T \boldsymbol{m}_j^- + b) + \eta_j \geq 1, \quad j = 1, \ldots, \ell^-, \\ & \xi_i, \eta_j \geq 1 \quad i = 1, \ldots, \ell^+ \quad j = 1, \ldots, \ell^-, \end{aligned} \tag{A.9}$$

here, $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are the hinge losses from equation (A.6) for the individual bags [32, 77, 81, 82]. C_1 and C_2 are added to be able to punish false positives and false negatives differently. This is done because in most MIL problems there is a class imbalance, where the number of negative bags can be a lot larger than the number of positive bags.

Using an LP solver, one can find the optimal solution [50]. The LP solver will force most components of $\boldsymbol{w}^*$ to zero. The magnitude of a component w_l^* corresponds to how far the lth instance is from the concept center. Thus the selected concepts can be written as $\{s(\boldsymbol{x}^l, \cdot) : l \in \mathcal{L}\}$. Here $\mathcal{L}$ is:

$$\mathcal{L} = \{l : |w_l^*| > 0\}, \tag{A.10}$$

or in other words, the index of nonzero entries in the vector $\boldsymbol{w}^*$. Then the label of a new bag $\boldsymbol{B}_i$ can be determined by Equation 3.4:

$$\hat{y}_i = \text{sign}\left(\sum_{l=1}^{N_l} w_l^* s(\boldsymbol{x}_l, \boldsymbol{B}_i) + b^*\right).$$

To summarize, this method starts by embedding the training data into the concept space $\mathbb{F}_{\mathcal{C}}$. Next, an LP is formulated to solve a two-class classification problem using a sparse SVM. And finally, the nonzero weights of the LP indicate the index of instances that represent potential concepts.

Appendix B

Prism

The rule induction algorithm Prism was designed to find if-then rules based on categorical data [52, 53]. This algorithm was originally developed for the creation of expert systems, where the conditions listed in the IF part of the rules are properties of the eyes, and the outcome is the best possible contact lens for those eyes [53, 83]. This appendix will explain the basic idea of Prism and then give the exact algorithm used by the proposed MIL classifier called Concept Rule Induction (CRI).

Prism is an example of a sequential covering algorithm. Sequential covering algorithms try to induce rules one by one, where each new rule covers a new part of the data set. The outcome of a rule is limited to a single option or class in the classification setting. This continues until all examples in the data set are covered by at least one rule in the rule set for a specific class. This is why CRI uses a one-against-all approach as these rules only apply to a single class.

Prism will be used on the concept bag matrix $\boldsymbol{C_B}$ as described in Equation 3.2. The rows of this matrix correspond to which concepts are in each bag $\boldsymbol{B}_i$ of the training set $\boldsymbol{B}$. The columns of this matrix correspond to different concepts $[c_1..c_{N_C}] \in \mathcal{C}$, where $\mathcal{C}$ is the set of N_C concepts found by the concept algorithm in section 3.1. The entries of $\boldsymbol{C_B}$ are binary. The $(i^{\text{th}}, k^{\text{th}})$ position corresponds to presence of concept c_k in bag $\boldsymbol{B}_i$.

In CRI the label sets of each bag are reduced to a two-class problem for each different class in a one-versus-all fashion: a target class q is chosen and then bag $\boldsymbol{B}_i$ is given the label $y_i = 1$ if the target class q appears in the label set of $\boldsymbol{B}_i$, $y_i = 0$ otherwise. Let $\boldsymbol{Y}_{\boldsymbol{B}}^q$ be the vector of all binary labels produced this way for the training set $\boldsymbol{B}$. As defined in section 2.3, $\boldsymbol{R}_q$ will be the output of the Prism algorithm for the target class q.

Rules in Prism are built using conditions on the output of the concept detection algorithm. Let $f_c(\boldsymbol{B}_i)$ be the condition that concept c appears in bag $\boldsymbol{B}_i$, meaning $f_c(\boldsymbol{B}_i) = \text{True}$. Let $\neg f_c(\boldsymbol{B}_i)$ be the opposite, $\neg f_c(\boldsymbol{B}_i) = True$ when the concept c does not appear in bag $\boldsymbol{B}_i$. Then define the following function *getBestCondition*$(\boldsymbol{C_B}, \boldsymbol{Y}_{\boldsymbol{B}}^q, f'_{c1}, f'_{c2})$, which gives back the best condition of the conditions f'_{c1} and f'_{c2} based on the concept bag matrix and the binary labels, see algorithm 1.

The conditions in *getBestCondition* are ranked first on the highest ratio of the number of lines that give the target class implication. Then if ratios are equal, the conditions are ranked on the highest number of target class implications. And lastly, if the number of target class implications is equal as well, the condition with the lowest amounts of false implications ($f(\boldsymbol{B}_i) = True \wedge y_i^q = False$) is chosen. This makes *getBestCondition* return the condition f'_C which is the largest cover of the target class while avoiding conditions that do contribute to the target class. If one

Algorithm 1: GetBestCondition function, compares two conditions on the concept bag matrix and its labels to see which condition is a better fit.

Input : The concept bag matrix $\boldsymbol{C_B}$, the binary label vector $\boldsymbol{Y_B^q}$, the previous best condition f'_{c1} and the condition to be tested f'_{c2}.
Output: The best of the two conditions f_{best}.

getBestCondition($\boldsymbol{C_B}, \boldsymbol{Y_B^q}, f'_{c1}, f'_{c2}$)

`// Get the number of implications; given the condition is true, does it imply the target label?`

let $I_{c1} = \sum_{\boldsymbol{B}_i}^{\boldsymbol{C_B}} [1 \text{ if } f'_{c1}(\boldsymbol{B}_i) = True \wedge y_i^q = True,\ 0 \text{ otherwise}]$
let $I_{c2} = \sum_{\boldsymbol{B}_i}^{\boldsymbol{C_B}} [1 \text{ if } f'_{c2}(\boldsymbol{B}_i) = True \wedge y_i^q = True,\ 0 \text{ otherwise}]$

`// Get the number of true conditions; how many times does the condition hold?`

let $H_{c1} = \sum_{\boldsymbol{B}_i}^{\boldsymbol{C_B}} [1 \text{ if } f'_{c1}(\boldsymbol{B}_i) = True,\ 0 \text{ otherwise}]$
let $H_{c2} = \sum_{\boldsymbol{B}_i}^{\boldsymbol{C_B}} [1 \text{ if } f'_{c2}(\boldsymbol{B}_i) = True,\ 0 \text{ otherwise}]$

`// Get the ratio between implications and number of true conditions`

let $P_{c1} = I_{c1} \div H_{c1}$ if $H_{c1} \neq 0,\ 0$ otherwise
let $P_{c2} = I_{c2} \div H_{c2}$ if $H_{c2} \neq 0,\ 0$ otherwise

if
  $(P_{C2} > P_{C1}) \vee$
  $(P_{C2} = P_{C1} \wedge I_{C2} > I_{C1}) \vee$
  $(P_{C2} = P_{C1} \wedge I_{C2} = I_{C1} \wedge H_{c2} < H_{c1})$
then
  return f'_{c2}
else
  return f'_{c1}
end

of the conditions is equal to the empty condition $\emptyset$, then it will return the other new condition.

A rule $\boldsymbol{r} \in \boldsymbol{R}_q$ is made up several conditions f'_c. Because this paper only uses simplified propositional logic, each condition is *True*. Adding a new condition to a rule $\boldsymbol{r}$ does not always add more coverage to the target class q. At that point, the rule is done, and no further conditions can be added to improve the rule. Prism moves on to the next rule by removing the rows that are covered by rule $\boldsymbol{r}$ from the concept bag matrix. To check if a new condition f'_c adds coverage a rule $\boldsymbol{r}$ the function $\mathit{increasesCoverage}(\boldsymbol{C_B}, \boldsymbol{r}, f'_c)$ is used. The *increasesCoverage* function return *True* if more rows from $\boldsymbol{C_B}$) are covered by adding the condition to the rule, *False* otherwise.

Algorithm 2: Prism, a sequential covering rule induction algorithm, adapted to be used in combination with Concept Rule Induction.

Input : The concept bag matrix $\boldsymbol{C_B}$ and the binary one-versus-all bag label vector $\boldsymbol{Y_B^q}$ for target class q.

Output: The rule set $\boldsymbol{R}_q$, which contains a set of simplified propositional logic rules which can be used to classify new bags, given the presence of concepts of the concept set $\mathcal{C}$.

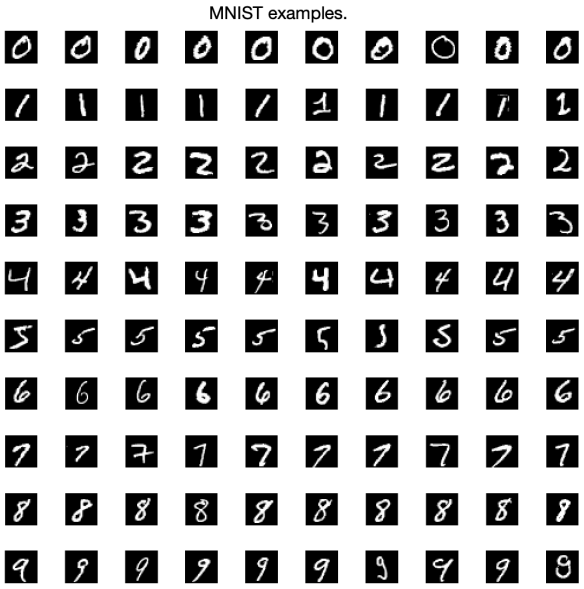
```
Prism (C_B, Y_B^q)
let R_q = ∅
while ∃ y_i ∈ Y_B^q | y_i = 1 do
    let C' = C
    let r_new = ∅
    while C' ≠ ∅ do
        let f_best = ∅
        foreach c ∈ C' do
            foreach f'_c ∈ [f_c, ¬f_c] do
                if increasesCoverage(C_B, r_new, f'_c) then
                    f_best = getBestCondition(C_B, Y_B^q, f_best, f'_c)
                end
            end
        end
        if f_best ≠ ∅ then
            add f_best to r_new
            remove c ∈ f_best from C'
        else
            break
        end
    end
    if r_new ≠ ∅ then
        add r_new to R_q
        remove rows B_i from C_B and Y_B^q that are covered by r_new
    else
        // Adding more rules would decrease coverage
        break
    end
end
return R_q
```

At this point, the Prism algorithm can be described in algorithm 2. The original Prism algorithm was primarily used for categorical data [53, 54], so it had to check all the options in each category separately. In this paper categories correspond to concepts, so only Prism only has to check whether to concept appears in a bag or not.

Appendix C

Multiple Instance Learning MNIST toy data creation

In this appendix, three MIL toy problems are discussed. The first problem is known as the trivial MIL problem. The other two are based on the MNIST data set. MNIST stands for the Modified data set from the National Institute of Standards and Technology [3]. This data set contains handwritten numbers. Some example images can be seen in Figure C.1.

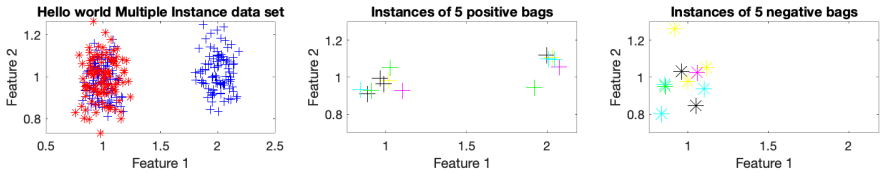


`Figure C.1:` MNIST examples pictures[3]. Each row corresponds to ten examples from one of the classes from zero to nine.

The MNIST data set is not a MIL data set. Each image is 28 by 28 pixels, or 784 pixels, with a single gray-scale value from 0 to 255. To transform this single feature vector per image into a MIL data set two approaches are taken. These two data sets are called pos-MILMNIST and patch-MILMNIST and their exact transformations are described below.

C.1 A trivial MIL toy problem

The *hello world example* of multiple instance data sets which in this paper is called the trivial problem, still uses two concepts. There are two classes in the data set. The first, and positive, class always contains instances from both concepts A and B. The second, and negative class, contains concepts from only concept A. This means only the line between concept A and B has to be found to classify all data correctly. This data set is visualized in Figure C.2.

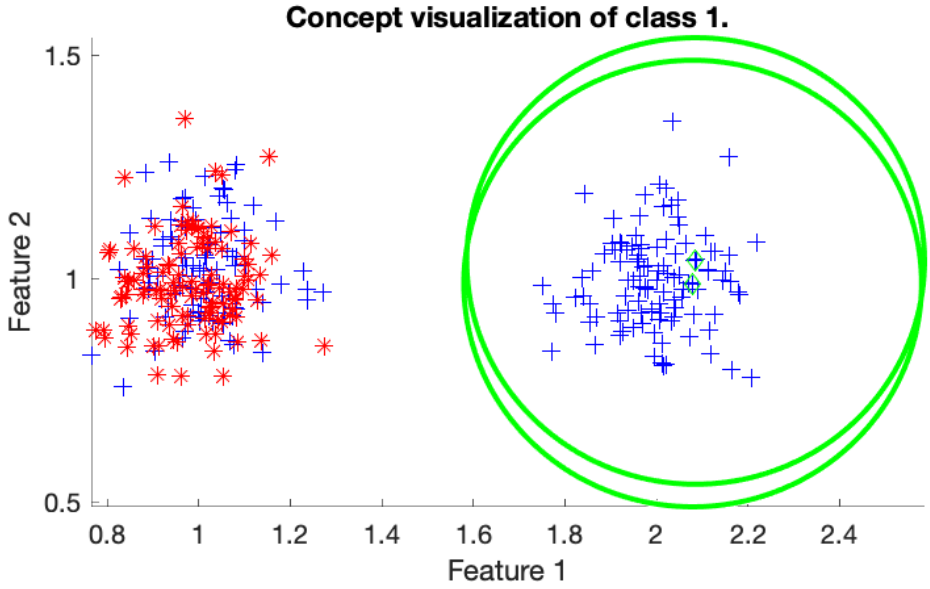


`Figure C.2:` The simple hello world MI data set consists of 50 positive and 50 negative bags. On the left is the entire data set with positive bags in blue, and negative bags in red. In the middle: 5 different bags in different colors which are positive. On the right: are 5 different bags in different colors which are negative.

A rule set that can classify the positive class would only need to check for the appearance of concept B in a bag. In Figure C.3, the concepts that the MILES algorithm produces can be seen. The Prism algorithm produces the following ruleset:

$$\boldsymbol{R}_{\text{trivial}} = \Big\{ r_1 : f_1$$

Note that while in the example concept B is given the name B, the algorithm can only each non-zero weight and assign a number based on chronological order in the training set to the concept. Nevertheless, this is the expected rule set.



`Figure C.3:` MILES produced concepts. The chosen instances have a small green circle over them, the larger green circles indicate the decision boundary of the concept classifiers. Note, though a single concept should have been enough, the MILES classifier still produced two concepts.

C.2 A positional-based sampling from MNIST: pos-MILMNIST

A low-dimensional transformation on MNIST can be done by averaging the grayscale values of small patches together with the positional coordinates of those patches. If this value is black, it can be left out of the representative bag. This results in instances of just two dimensions, the x and y positional values. This data set will be called pos-MILMNIST, which stands for positional - Multiple Instance Learning MNIST. The precise transformation is as follows:

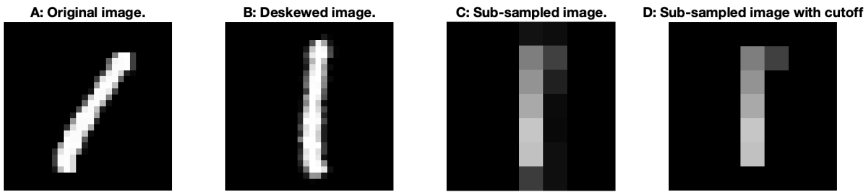
1. A common pre-process step of deskewing the MNIST images is done. This process tries to straighten the images because some handwritten digits are more slanted than others. The process is the same as the authors described in the original MNIST paper [3].

2. The images are sub-sampled to reduce the total amount of instances. The images are divided up into a 7 by 7 grid of 4 by 4 pixels. This creates a maximum of 49 instances per image. For the pixel value, the average of the 4 by 4 pixels is taken.

3. Using a threshold, very dark instances are excluded from the bag. These very dark instances don't visually contribute much so they are ignored to further reduce the average amount of instances per image.

4. Every image is seen as a bag, with the digit number being the bag label. The x, y positions of the sub-sampling are the features.

This creates a MIL data set that includes the spatial coordinates of pixel values, which will result in being able to compare rule sets of different numbers. And because the x and y values of the sub-sample are features, it is straightforward to visualize the rule sets as well. The transformation for a single digit is shown in Figure C.4.



`Figure C.4:` MNIST examples [3]. The transformation from MNIST to a MIL data set. A. The original image is a handwritten 'one'. B. The deskewed image. C. The sub-sampling of the deskewed image. And D. The leftover instances after the background removal.

C.3 Random sampling patches from MNIST: patch-MILMNIST

A more visual approach to generate a MIL MNIST data set is to take a set of random patches from the MNIST images. The idea behind this approach is that these patches will contain parts of numbers, such as lines, circles, and corners. A one, for example, should only contain lines, while a zero should only contain circles or part of circles.

Let's call this data set patch-MILMNIST, as a continuation of MILMNIST. The data set is created by executing the following steps:

1. To start, like the pos-MILMNIST data set, the images are deskewed as described in the original MNIST paper [3]. This makes the problem slightly easier to solve, reducing the number of generated PRISM rules.

2. Then, select some number of random patches of the same size. In all following examples patches of 8 by 8 pixels are taken for a total instance feature size of 64 dimensions. Each instance feature corresponds to the gray-scale value of that pixel.

3. Each image from MNIST is sampled for all patches. A bag corresponds to a single image, and its instances are then 8 by 8 pixels from each patch.

4. Each instance that only contains very dark pixels (pixel gray-scale values below 25) is thrown away. These patch instances are mostly dark and don't contribute visually much to the classification.

In this procedure, there is no need to normalize the values as with pos-MILMNIST. Because the only features used here are gray-scale values, which are all in the same range of 0 to 255. An example of instances generated from a handwritten digit can be seen in Figure C.5.

`Figure C.5:` The thirteen instances generated by taking twenty random patches from an MNIST image. On the left, an example image of a handwritten nine is shown, with the patches plotted on the number. All smaller pictures on the right are the generated instances from these patches. Only thirteen instances are generated because the other instances contain an average gray-scale value below the cut-off point. The cut-off instances are almost entirely dark squares.

Appendix D

Parameter optimization for the used concept detectors

In order to find concepts, both MILES and k-means have parameters that have to be set correctly. In this appendix, the parameters of the concept classifiers are examined in their behavior.

MILES advertises only a single parameter, the scaling factor σ, but there is actually a hidden parameter, which is the regularization parameter λ of the sparse classifier used in MILES [32]. This parameter has an impact on the number of concepts found, and it might even result in not finding any concepts at all. This appendix will look at the regularization parameter λ and the scaling factorσ from MILES first, after which the amount of clusters parameter k from k-means is examined.

D.1 MILES, the regularization parameter

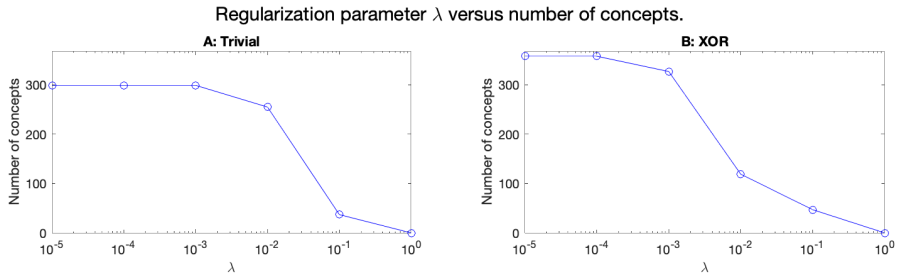
The regularization parameter λ from MILES depends on the implementation of the sparse linear classifier. MILES is formulated as a linear program. For linear programs, many solvers exist. This paper used the SLEP package, where SLEP stands for Sparse Learning with Efficient Projections [84]. This package was chosen because it achieved good and fast results on the data sets in this paper, furthermore, it was free. From this package, the sparse logistic classifier was chosen. This classifier minimizes the following loss with regularization parameter λ:

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{N_i} \log\left(1 + \exp\left(-y_i(\boldsymbol{w}^T s(\cdot, \boldsymbol{B}_i) + b\right)\right) + \lambda|\boldsymbol{w}|, \tag{D.1}$$

where y_i is the label of bag $\boldsymbol{B}_i$ and $s(\cdot, \boldsymbol{B}_i)$ is the MILES similarity embedding vector of that bag on the training instances. N_i is the total number of training bags. And $\boldsymbol{w}$ and b are the optimization variables. T is the transpose operator in this equation, so that the weight vector $\boldsymbol{w}$ can be used for the inner product multiplication with the embedding vector. The regularization term $\lambda|\boldsymbol{w}|$ uses the L_1-norm of the weight vector $\boldsymbol{w}$. Note that $w_l \in \boldsymbol{w}$ corresponds to training instance $\boldsymbol{x}_l$.

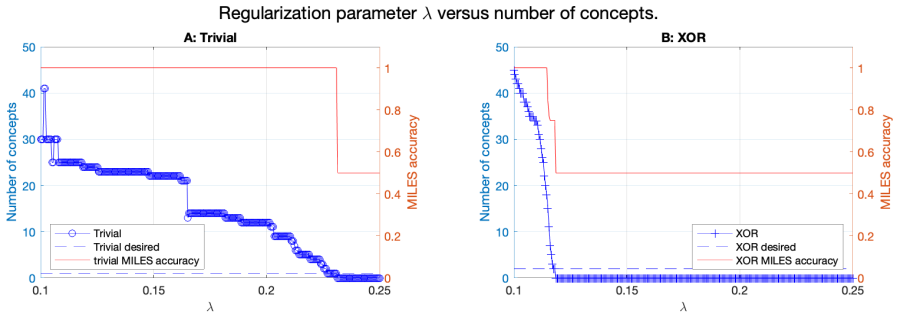
If λ is higher, the norm $\boldsymbol{w}$ has a larger influence on the minimization term. At some point, the regularization term influence will get so high that it is no longer possible to find a good minimization of Equation D.1. For the toy example it is easy the define a good σ as it is good enough to capture instances in one of the clusters, so let's set it to $\sigma = 0.5$. First, λ is varied exponentially in Figure D.1. It is no longer possible to find any concepts after $\lambda > 1$. Here, all weights $w_l \in \boldsymbol{w}$ are zero, meaning

that no instances were selected as possible concepts. This results in a useless MILES classifier, and it is not possible to continue with any rule induction.



`Figure D.1:` The number of concepts versus the regularization parameter for the trivial and XOR miles problem.

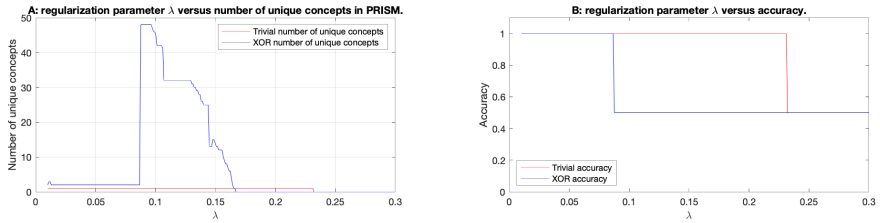
It seems to be possible to find a λ that results in finding the desired number of concepts. After zooming in on Figure D.1 in Figure D.2, these values can be found. These values are different depending on the problem. For the trivial toy example λ can be a bit higher with a value around $\lambda_{\text{trivial}} = 0.232$ gives 1 concept and for the XOR example a value around $\lambda_{\text{xor}} = 0.1336$ gives 2 concepts.



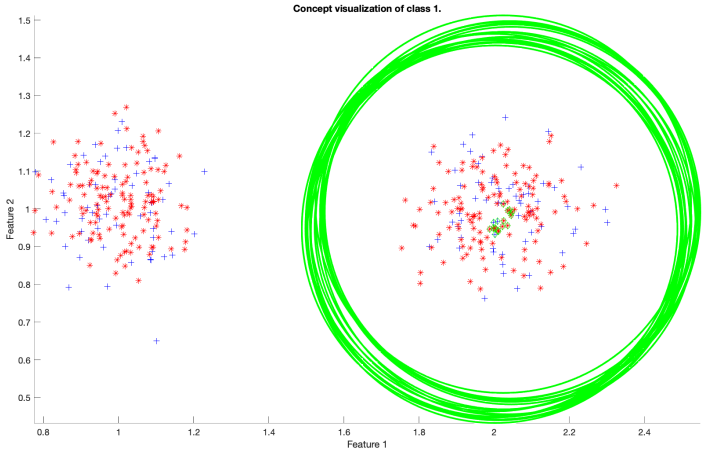
`Figure D.2:` The number of concepts versus the regularization parameter zoomed in, with the desired number of concepts for both toy problems. On the right axis, the MILES performance for both problems can be seen.

Another way of judging the influence of λ is by looking at the rule induction and classification results. This can be seen in Figure D.3. For the rule induction, it can be seen that something interesting happens around $\lambda > 0.1$. Here the number of unique concepts in the rule set suddenly shoots up. It seems this is because the optimizer at this point cannot find a good minimal solution, and it seems like the MILES only choices concepts in one of the two concepts blobs. In Figure D.4, the concepts that are found at $\lambda = 0.13$ are visualized. Because the concepts are only from one side of the XOR problem identified, the concept stage does not capture the problem correctly. The rule "garbage in, garbage out" applies here. If the rule induction algorithm is fed non-separable data, it won't be able to classify accurately.

However, what is good is that for lower values of λ the Prism algorithm seems to be able to prune the many concepts down to two rules and concepts for the XOR problem and only one for the trivial problem. Thus, the regularization parameter λ should be set so that MILES can find a minimal solution that has non-zero weights, and then the rule induction algorithm can prune additional concepts.



`Figure D.3:` The influence of regularization parameter λ on A: the number of unique concepts. And B, the classification accuracy.



`Figure D.4:` The concepts found for $\lambda = 0.13$ on the MIL XOR problem. Note that class 1 actually has two concepts here, but only one is found multiple times. This seems to be due to that the regularization term forces the solver to give back an undesired solution.

D.2 MILES, the scaling parameter

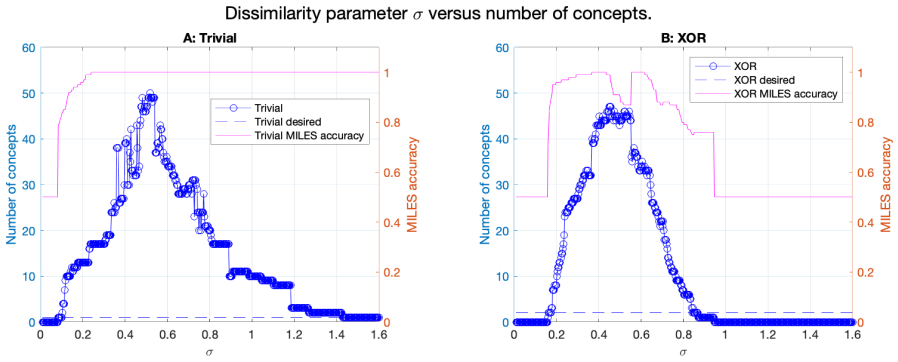
The other parameter of the pipeline is the MILES scaling σ. This parameter is from a similarity metric from the MIL diverse density framework [49] and is used during the MILES embedding step in the following formula:

$$s(\boldsymbol{x}^k, \boldsymbol{B}_i) = \max_j \exp\left(-\frac{||\boldsymbol{x}_{ij} - \boldsymbol{x}^k||^2}{\sigma^2}\right), \tag{D.2}$$

where $s(\boldsymbol{x}^k, \boldsymbol{B}_i)$ is the embedding of instance $\boldsymbol{x}^k$ on bag $\boldsymbol{B}_i$. This metric is also called the most-likely-cause estimator. As can be observed, σ acts as a scaling parameter, and needs to be set depending on the distances between instances in the data set.

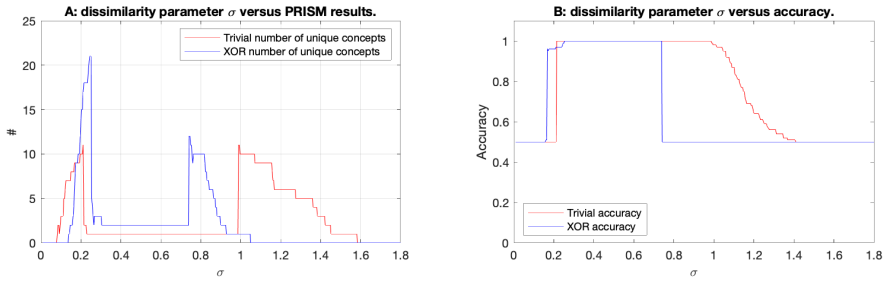
Like in the previous section, let's first look at the number of concepts the MILES classifier produces depending on different values of σ. Note that the standard deviation of the Gaussian concepts in both the trivial problem and the xor problem is have a rather small standard deviation. Setting $\sigma = 0.5$ would capture the majority of points from these distributions. In Figure D.5, it can indeed be seen that the number of concepts is the largest around 0.5 for both problems.

Then, when looking at the rule induction and classification accuracy, there should be parabolic behavior. When σ is too small, one or two concepts should not be enough to entirely capture the toy problems' behavior, thus more concepts or rules would be needed during rule induction. When σ is just right, only the minimal amount



`Figure D.5:` The number of non-zero weights, or in other words concepts, versus the MILES parameter σ on the left. On the right axis is the direct MILES accuracy. Note that for the XOR problem MILES does not at any point reaches an accuracy of 1.

of concepts for the toy problems are needed. And, when σ is too large it will start overlapping with the other Gaussian concept, incorrectly classifying the negative class as positive. This results in using more concepts in rules to classify correctly. This behavior can be fully seen in Figure D.6.



`Figure D.6:` The influence of MILES parameter σ on: A: the number of unique concepts in the rule set and the number of unique concepts. And B, the classification accuracy of PRISM.

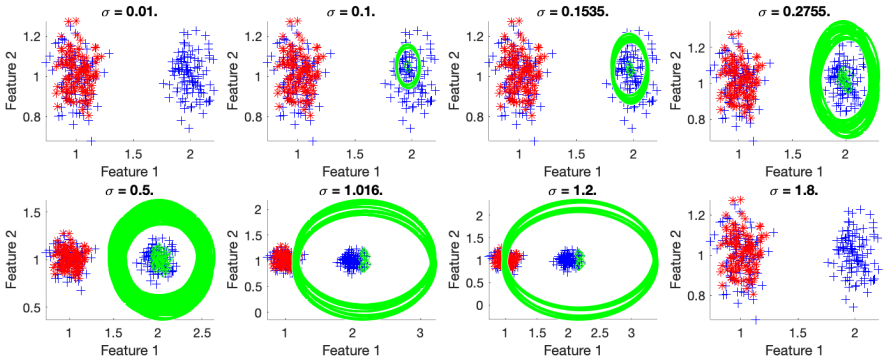
It seems that if σ is set too small, then PRISM cannot find good rules that actually classify correctly. In the trivial problem in Figure D.6, it can clearly be seen that if σ is too large it will overlap with the negative class concept. In Figure D.7, this behaviour can be seen. In this case, it might be worth separating the concept classifier MILES σ from the concept decision boundary from Equation 3.5.

Thus it seems that the optimal value of σ is highly dependent on the data set, but if set correctly in the valley as seen inFigure D.6 Prism can handle the output of MILES as a concept classifier.

D.2.1 Setting the scaling per instance

The MILES parameter σ is dependent on the data set according to the previous sections. So perhaps this parameter can be initialized automatically depending on the data set behavior. Because instances are chosen as the concept centers, it seems that only the local behavior of data would need to be taken into account.

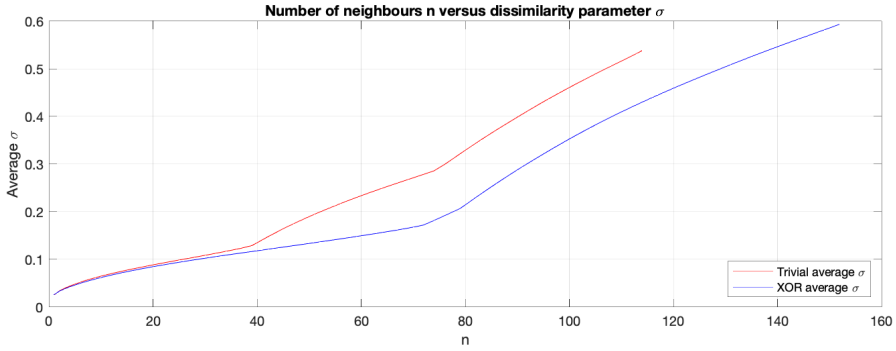
One way is by looking at the distances from the n nearest neighbors. Let $\boldsymbol{X}$ be the set of all instances. The parameter σ_k can then be estimated for a specific instance $\boldsymbol{x}^k \in \boldsymbol{X}$ by calculating the pair-wise euclidean distances between all instances $\boldsymbol{X}$.



`Figure D.7:` Concepts highlighted in green with their decision boundary for different values of the MILES parameter σ. For $\sigma = 0.01$ and $\sigma = 1.8$ no concepts are found, meaning the MILES optimiser only has zero values for weights. For $\sigma = 1.016$ and $\sigma = 1.2$, the size of the concept classifiers starts to overlap with the negative Gaussian concept, meaning classification performance starts plummeting.

Next, sort these distances and take the average distance from the n nearest neighbors. σ_k can then be plugged in Equation D.2 for every instance $\boldsymbol{x}^k$.

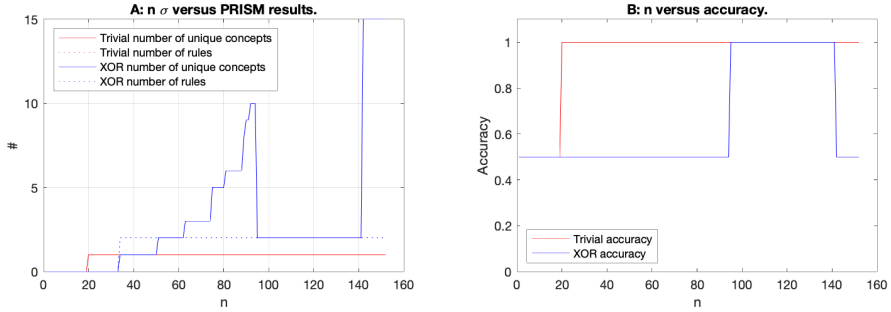
Now there is a separate scaling factorσ_k for each instance $\boldsymbol{x}^k$. By averaging the scaling factors for different amounts of neighbors it is possible to see how they relate. It is expected that the average value would increase if more neighbors are added because more points in concepts are taken into account. This can be seen in Figure D.8.



`Figure D.8:` The average value of σ as a result of the number of neighbours n by setting σ_k for each instance $\boldsymbol{x}^k$ individually. Note that the number of neighbors can only be set to the total amount of instances.

The effect can also be seen in the number of rules and concepts generated by PRISM and the classification results. See Figure D.9. This is similar to Figure D.6, where parabolic behavior occurs for the XOR problem.

Unfortunately, using this local behavior around the instances seems to not have the desired impact. The best results from Figure D.9 seem to occur when almost the entire data set is taken into account, losing the local behavior of setting a σ_k for each instance $\boldsymbol{x}_k$.



`Figure D.9:` The influence of the number of neighbors n when setting σ_k on: A: the number of unique concepts in the rule set and the number of unique concepts. And B, the classification accuracy.

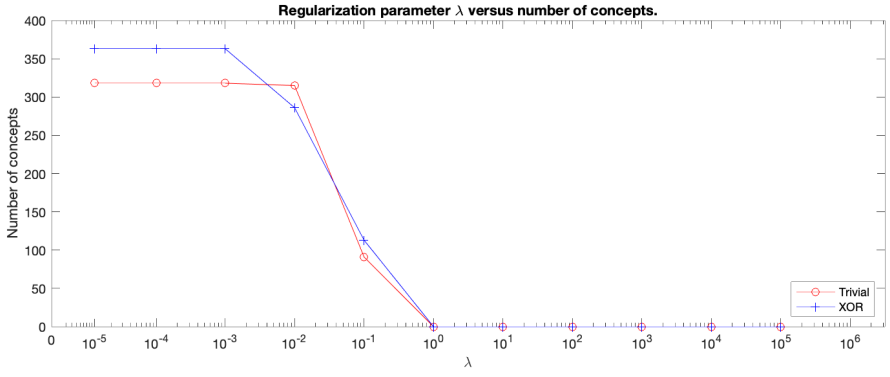
D.2.2 Binary similarity MILES

Another way to construct a concept classifier would be to stick with MILES but change the similarity metric from Equation D.2 to be more in line with the concept definition from Equation 3.5. Thus $s(\boldsymbol{x}^k, \boldsymbol{B}_i)$ becomes

$$s(\boldsymbol{x}^k, \boldsymbol{B}_i) = \max_j \begin{cases} 1 & \exists\ \boldsymbol{x}_{ij} \in \boldsymbol{B}_i : \left\|\boldsymbol{x}^k - \boldsymbol{x}_{ij}\right\| < \sigma, \\ 0 & \text{otherwise.} \end{cases}, \tag{D.3}$$

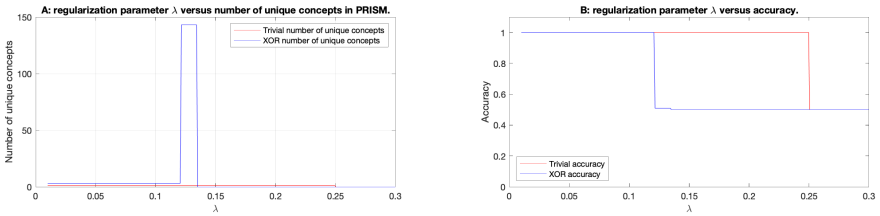
The advantage of this approach is that similarity now matches the concept definitions. The similarity is now more strict, which should prevent small concepts such as those from Figure D.7. These small concepts clearly don't capture the entire don't capture the actual concept. This approach still has the two other parameters from MILES so let's look at σ and λ for the similarity metric from Equation D.3.

The regularization parameter λ has slightly less influence here if it is low enough compared to those of regular MILES in Figure D.2. In Figure D.10, similar behavior to normale MILES can be found, but if when zoomed in as for Figure D.11 it seems that the new similarity metric is stable for larger values of λ for the same problems.



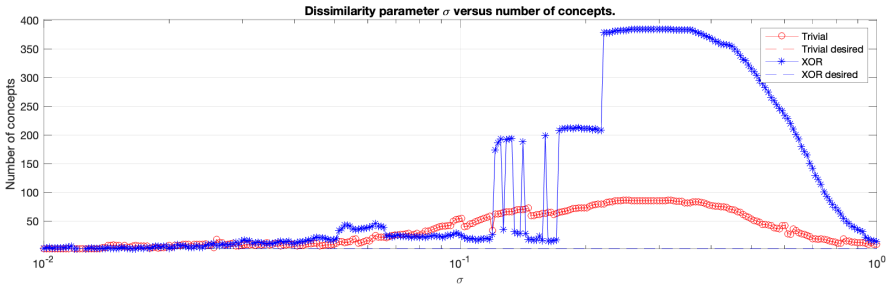
`Figure D.10:` The number of concepts versus the regularization parameter λ for the trivial and XOR miles problem. Using the strict similarity for MILES from Equation D.3.

In Figure D.11A there is a large peak of the number of concepts for the XOR-problem that PRISM uses during classification. It seems that at that point the concepts again appear only on one of the two actual concept blobs. It seems again though that there is no clear correlation for the regularization parameter, and that has to be experimentally determined to which point it does not revert to the trivial null solution.



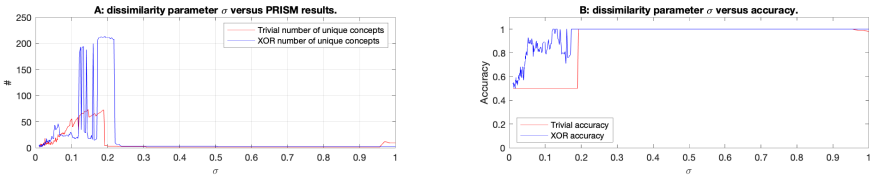
`Figure D.11:` The influence of regularization parameter λ on A: the number of unique concepts. And B, the classification accuracy. Using the strict similarity for MILES from Equation D.3.

For the similarity parameter σ from Equation D.3. it seems that taking a value that is too low now increases the number of concepts, but as soon as σ is large enough to cover most of the concept blobs from both toy problems, the number of concepts required by PRISM goes down again. This can be seen in Figure D.12, where roughly a $\sigma = 0.5$ would be ideal. Though it seems that more concepts are still identified when that number is reached (about 50). This might be due to that the regularization parameter here is not high enough.



`Figure D.12:` The number of concepts versus the MILES parameter σ for the trivial and XOR miles problem. Using the strict similarity for MILES from Equation D.3.

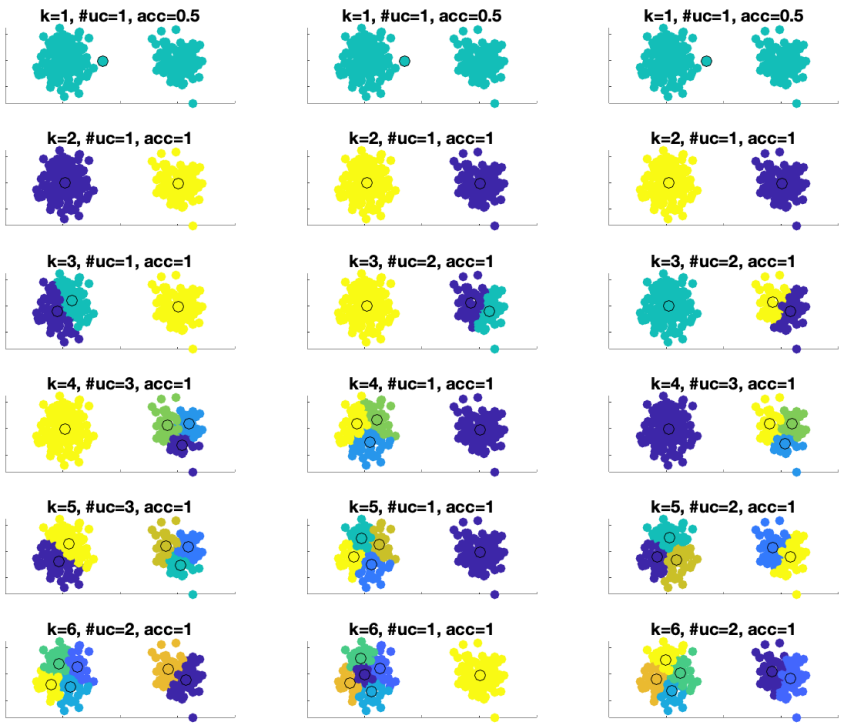
In Figure D.13 it can be seen that when σ approaches 1, the accuracy slightly dips again for the trivial toy problem. This is because at that point instances from negative bags enter the positive concept. When compared to the original MILES concept detection algorithm it does seem that this binary similarity performance is slightly better in the rule induction step. But this approach does lose the gaussian behavior of MILES and many more concepts are produced than with the regular similarity metric.



`Figure D.13:` The influence of MILES parameter σ on A: the number of unique concepts. And B, the classification accuracy. Using the strict similarity for MILES from Equation D.3.

D.3 k-means, the number of clusters

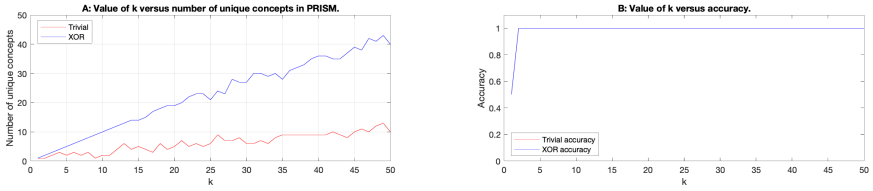
The k-means algorithm uses random initialization for its k cluster centers. This can have an effect on finding different concept classifiers. In Figure D.14 some different values of k are plotted for the trivial problem. Note that in this figure for the trivial problem the number of unique concepts used by PRISM is equal to the number of clusters on the right blob. It should also be remarked that, while for the trivial problem only a single concept is needed, two concepts are still required for meaningful classification, because the k-means clustering algorithm has no cut-off distance and will always put any point into the closest cluster.



`Figure D.14:` The influence of different random initialization with the same k on the concepts for the Trivial problems. The value of k is shown in each subplot. *#uc* stands for the number of unique concepts used by PRISM. And *acc* stands for the accuracy of the PRISM algorithm. Cluster centers, which also act as concepts, are marked by a black circle. Notice that the number of unique concepts always equals the number of clusters for the trivial problem concepts in the right blob.

Using k-means does introduce the k parameter, but this is a *single* parameter compared to *two* of the MILES concept detection. The toy examples from Appendix C are defined as two clusters, so $k = 2$ should be the optimal value. In Figure D.15, it can be seen that if k increases more concepts are needed by PRISM to get the correct classification. This is because the division of the blob concepts increases if k increases. Though at a certain point, some clusters will only contain instances from negative bags, which is why the line in Figure D.15A is not entirely linear to k.

The classification accuracy is equal to 1 after reaching the required two concepts to differentiate between the two classes.



`Figure D.15:` The influence of k-means parameter k on: A: the number of unique concepts in the rule set and the number of unique concepts. And B, the classification accuracy, which overlaps for both problems here.

Thus, having some knowledge about the data set helps to determine the perfect value for k. Additionally, it is important to do several initializations of the clustering algorithm to see if there is a weak initialization. If using k-means has a significant variance of performance for different initializations in CRI, then there are no clearly defined clusters in the data set. Then a more fuzzy approach like MILES would be a better choice as a concept detector.