Boolean kernels for rule based interpretation of support vector machines

Boolean kernels for rule based interpretation of Support Vector Machines

Mirko Polato and Fabio Aiolli

University of Padova - Department of Mathematics
Via Trieste, 63, 35121 Padova - Italy

Abstract

Machine learning started as an academic-oriented domain, but nowadays it is becoming more and more widespread across diverse domains, such as retail, healthcare, finance, and many more. This non-academic face of machine learning creates a new set of challenges. The usage of such complex methods by non-expert users has increased the need for interpretable models. To this end, in this paper we propose an approach for extracting explanation rules from support vector machines. The core idea is based on using kernels with feature spaces composed by logical propositions. On top of that, a searching algorithm tries to retrieve the most relevant features/rules that can be used to explain the trained model. Experiments on both categorical and real-valued datasets show the effectiveness of the proposed approach.

1. Introduction

Machine learning (ML) is becoming increasingly ubiquitous. Nowadays, ML models are used for almost everything, from predicting the stock price of a company to detecting object in an image. However, when it comes to apply ML by non expert users the black box nature of most of ML methods can become an obstacle. The lack of interpretability of many machine learning methods, e.g., kernel machines and (deep) neural networks, makes hard their application in scenarios in which explanations are as important as the prediction quality, for example, support systems for physicians and recommender systems. Interpretability is also key to winning trust in algorithms that try to improve upon human judgement, instead of just automating it.

The need of explanations is also theme of the Article 22.1 of the General Data Protection Regulation which states that: *"The data subject shall have the right not to be subject to a decision based solely on automated processing"*[1].

In the past, some efforts have been devoted in order to alleviate this black-box nature of ML models [2, 3]. In this work we focus on interpreting kernel machines, in particular SVM. In the literature [2], most of the proposed methods for extracting explanation rules from SVM are based on the definition of regions in the input space that are then converted into *if-then-else* rules. Here, we propose a different approach which works directly in the feature space. Specifically, by means of Boolean kernels, which have shown state-of-the-art performance in binary classification tasks [4, 5], the data are mapped onto an easy-to-interpret feature space, and in such space an SVM is trained (BK-SVM). Since the feature space of a BK-SVM is composed of Boolean rules, it is possible to give a human-readable interpretation of the solution of the SVM by extracting the most influential rules in the decision.

Besides the application to categorical datasets, we show the effectiveness of the method on a real-valued dataset. The application of Boolean kernels on such data is made possible by a discretization pre-processing step. Applied on the UCI [6] Breast Cancer Wisconsin Data Set, we compare the interpretation rule extracted by our proposed algorithm w.r.t. the rule set extracted by state-of-the-art rule extraction methods.

So, the main contribution of this work is five-fold:

- first we present a new Boolean kernel which creates a feature space made of (potentially) all possible monotone DNF formulas over the input variables;
- we propose an algorithm for extracting from the BK-SVM the most relevant rule which can be used to interpret the solution of the support vector machine;
- we describe a method for applying Boolean kernels to non binary datasets;
- we show a thoroughly set of experiments on both artificial and real-world datasets. We analyze strength and weaknesses of the proposed method on real-world categorical and real-valued dataset;
- a comparison with rule extraction technique is provided.

The remainder of the paper is structured as follows: Section 2 presents the necessary background about Boolean kernels; Section 3 describes the first main contribution, that is, the construction of a kernel in which the feature space is composed of potentially all mDNF formula over the input variables; Section 4 presents the algorithm for extracting interpretation rules from a SVM based on Boolean kernels; Finally, Section 5 shows all the performed experiments on both artificial and real-world datasets.

2. Background

Throughout the paper we consider binary valued datasets for binary classification tasks. Formally, $\mathcal{T} \equiv \{\mathbf{x}_i, y_i\}_{i=1}^{L}$ is a training set where $\forall\, i, \mathbf{x}_i \in \{0,1\}^n$ and $y_i \in \{+1, -1\}$. We refer to generic n-dimensional Boolean vectors with $\mathbf{x}$ and $\mathbf{z}$, and with the notation $\mathbf{x}^{\mathbf{b}}$ we indicate the component-wise exponentiation, i.e., $x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$. Finally, the notation $[\![\cdot]\!]$ represents the indicator function.

2.1. Boolean kernels

In the literature, the term Boolean kernel has been used as a synonym of the so-called DNF kernel [7]. However, in a recent work [4] a broader definition has been provided: Boolean kernels are kernel functions which take binary vectors as input and apply the dot-product in a feature space where each dimension represents a logical proposition over the input variables. The general idea behind Boolean kernels is depicted in Figure 1.

2.1.1. Monotone Conjunctive kernel

One of the simplest Boolean kernel is the monotone Conjunctive kernel (mC-kernel) [5]. As the name suggests, its feature space is composed by all conjunctions of exactly c different input variables, where c is an hyper-parameter. Hence, the mC-kernel of arity c between $\mathbf{x}$ and $\mathbf{z}$ computes the number of *true* conjunctions of c literals in common between $\mathbf{x}$ and $\mathbf{z}$. Formally, the embedding of the mC-kernel of arity c is given by $\boldsymbol{\phi}_{\wedge}^{c} : \mathbf{x} \mapsto (\boldsymbol{\phi}_{\wedge}^{(\mathbf{b})}(\mathbf{x}))_{\mathbf{b} \in \mathbb{B}_c}$, where $\mathbb{B}_c = \{\mathbf{b} \in \{0,1\}^n \mid \|\mathbf{b}\|_1 = c\}$, and $\boldsymbol{\phi}_{\wedge}^{(\mathbf{b})}(\mathbf{x}) = \mathbf{x}^{\mathbf{b}}$. The dimension of the resulting feature space is $\binom{n}{c}$. Thus, the mC-kernel of arity c is computed by $\kappa_{\wedge}^{c}(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{c}$.

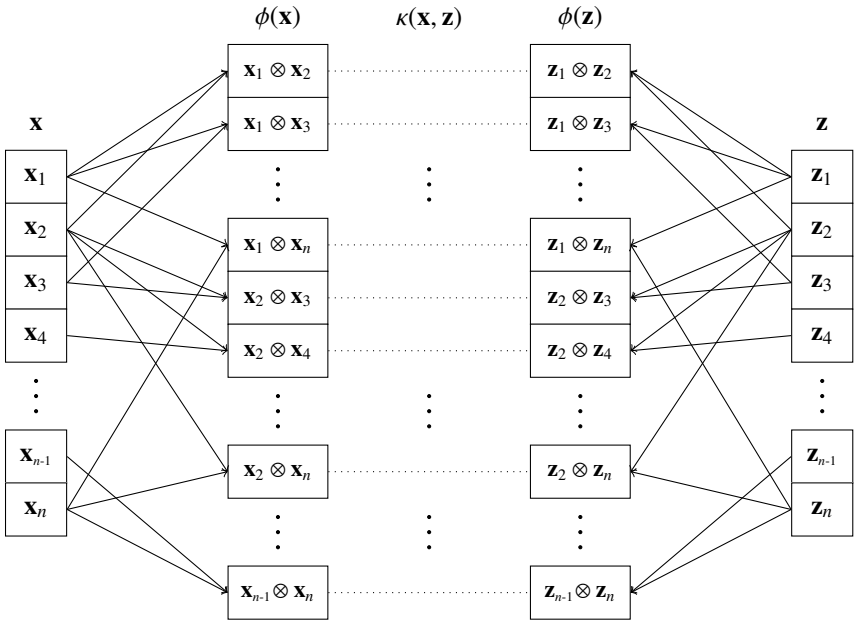


Figure 1: Depiction of a generic Boolean (operator $\otimes$) kernel of arity 2: firstly the input vectors are mapped into the feature space which is formed by all formulas with arity 2 (without repetition). Then, the kernel is computed by "matching" (dotted lines) the corresponding features. Arrows from the input vectors to the feature vectors indicate when a variable influences the formula.

It is worth to notice that the mC-kernel generalizes the linear kernel, in fact, by fixing $c = 1$ we obtain $\kappa^1_\wedge(\mathbf{x}, \mathbf{z}) = \binom{\langle \mathbf{x}, \mathbf{z} \rangle}{1} = \langle \mathbf{x}, \mathbf{z} \rangle = \kappa_{LIN}(\mathbf{x}, \mathbf{z})$.

2.1.2. Monotone Disjunctive kernel

Similarly to the mC-kernel, the monotone Disjunctive kernel (mD-kernel) [5] of arity d between $\mathbf{x}$ and $\mathbf{z}$ computes the number of *true* disjunctions of d literals in common between $\mathbf{x}$ and $\mathbf{z}$. Thus, the embedding of the mD-kernel is the same as the mC-kernel, however the logical interpretation is different since the combinations of variables represent disjunctions. Formally, the embedding of the mD-kernel of arity d is given by $\boldsymbol{\phi}^d_\vee : \mathbf{x} \mapsto (\boldsymbol{\phi}^{(\mathbf{b})}_\vee(\mathbf{x}))_{\mathbf{b} \in \mathbb{B}_d}$, with $\boldsymbol{\phi}^{(\mathbf{b})}_\vee(\mathbf{x}) = [\![\langle \mathbf{x}, \mathbf{b} \rangle > 0]\!]$. The corresponding mD-kernel is computed by

$$\kappa^d_\vee(\mathbf{x}, \mathbf{z}) = \binom{n}{d} - \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle}{d} - \binom{n - \langle \mathbf{z}, \mathbf{z} \rangle}{d} + \binom{n - \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{z}, \mathbf{z} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle}{d}. \quad (1)$$

The full derivation of Equation (1) is presented in [5].

2.1.3. Monotone Disjunctive Normal Form (DNF) kernel

A Disjunctive Normal Form (DNF) is a normalization of a logical formula that is represented by a disjunction of conjunctive clauses, e.g., $(x_1 \wedge x_3) \vee (x_5 \wedge x_6) \vee x_2$. The monotone DNF kernel (mDNF-kernel) [5] computes the dot-product of vectors in a feature space composed by monotone DNF formulas of the input variables, where the DNFs are composed by disjunction of exactly d conjunctive clauses made of c literals (c and d are hyper-parameters). For example, by fixing $d = 2$ and $c = 3$ a possible mDNF is $(x_1 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4)$. Its embedding map is the composition of the embedding maps of the mC-kernel and the mD-kernel, $\boldsymbol{\phi}^{d,c}_{\vee\wedge} : \mathbf{x} \mapsto \boldsymbol{\phi}^{d}_{\vee}(\boldsymbol{\phi}^{c}_{\wedge}(\mathbf{x}))$, hence the mDNF-kernel can be computed as the mD-kernel of degree d in the space formed by all conjunctions of degree c. Thus, computing the mDNF-kernel(d,c) is the same as Eq.(1) by substituting κ with the corresponding mC-kernels of degree c ($\kappa^c_\wedge$) and n with $\binom{n}{c}$, i.e., the dimension of the input space induced by $\phi^c_\wedge$. In the following we will refer to mDNF with at most D disjunctive clauses of conjunctive clauses of at most C literal with the notation mDNF(D,C).

3. The feature space of monotone DNFs

A shortcoming of the mDNF-kernel defined in Section 2.1.3 is that the mDNF formulas have a fixed form, that is, they are composed by disjunctions of d conjunctive clauses made of c literals. In order to overcome this limitation, we first create a feature space composed by all conjunctions up to a certain arity C, by concatenating the feature spaces of $\kappa^c_\wedge$ for $c \in [1, C]$, that is $\phi^C_\Sigma(\mathbf{x}) = (\phi^1_\wedge(\mathbf{x}), \phi^2_\wedge(\mathbf{x}), \ldots, \phi^C_\wedge(\mathbf{x}))$. The corresponding kernel can be implicitly computed [8] by $\kappa^C_\Sigma(\mathbf{x}, \mathbf{z}) = \sum_{c=1}^{C} \kappa^c_\wedge(\mathbf{x}, \mathbf{z})$. Now, by composing ϕ^C_Σ with $\phi^d_\vee$ (for some d) we obtain a feature space constituted of all possible mDNFs made of d conjunctive clauses of at most C literals. This kernel can be calculated by replacing $\langle \mathbf{x}, \mathbf{z} \rangle$ (i.e., the linear kernel) with $\kappa^C_\Sigma(\mathbf{x}, \mathbf{z})$ and n with $\sum_{c=1}^{C} \binom{n}{c}$ in Eq. (1). Finally, by summing up all these kernels with $d \in [1, D]$ we obtain a kernel with a feature space composed of all possible mDNF formulas with at most D conjunctive clauses of at most C literals. Formally,

$$\kappa^{D,C}_*(\mathbf{x}, \mathbf{z}) = \sum_{d=1}^{C} \kappa^d_\vee(\phi^C_\Sigma(\mathbf{x}), \phi^C_\Sigma(\mathbf{z})),$$

which is a valid positive semi-definite kernel because it is a result of closure properties [8].

4. Interpreting BK-SVM

One of the biggest advantages of using Boolean kernels is that the features in the embedding space are easy to interpret, and this characteristic can be leveraged to explain the solution of a kernel machine, e.g., SVM. In particular, the most influential features (i.e., logical rules) in the solution can be extracted in order to provide a human-readable interpretation of the decision. From the Representer Theorem [8] we know that the solution of an SVM can be written as $\mathbf{w} = \sum_{i \in \mathcal{S}} y_i \alpha_i \phi(\mathbf{x}_i)$, where $\mathcal{S}$ is the set of support vector indexes, and $\alpha_i \geq 0$ are the contributions of the support vectors to the solution. Hence, the weight associated to a feature f, i.e., a Boolean rule, inside the feature space induced by ϕ can be calculated by:

$$w_f = \sum_{i \in \mathcal{S}} y_i \alpha_i \phi_f(\mathbf{x}_i) = \sum_{i \in \mathcal{S}} y_i \alpha_i [\![f(\mathbf{x}_i)]\!] = \sum_{i \in \mathcal{S}^+} \alpha_i [\![f(\mathbf{x}_i)]\!] - \sum_{i \in \mathcal{S}^-} \alpha_i [\![f(\mathbf{x}_i)]\!], \qquad (2)$$

where $\mathcal{S}^+$ (resp. $\mathcal{S}^-$) is the set of positive (resp. negative) support vector indexes, and $\sum_{i \in \mathcal{S}^+} \alpha_i = \sum_{i \in \mathcal{S}^-} \alpha_i$. Our goal is to find the formula f such to maximize the value w_f. Since all α_i are positives and $\sum_i y_i \alpha_i = 0$, then this problem can be reduced to the one of finding the formula such that $w_f = \sum_{i \in \mathcal{S}^+} \alpha_i$. It is easy to show that, if (*i*) the set is linearly separable, (*ii*) the target concept is defined by a formula g, and (*iii*) the feature space contains g, then $g = \text{argmax}_f \; w_f = \sum_{i \in \mathcal{S}^+} \alpha_i$. It is noteworthy that w_f is maximized for every f consistent with the support vectors, and hence the best formula could not be unique. This is due to the fact that the target function generates the labels, then $\delta_f(\mathbf{x}_i) = 1 \Leftrightarrow y_i = +1$. Moreover, any other formula f for which it holds has $w_f = w_g$. In the case of non-separability, finding the rule that maximizes the value of w_f is still a good heuristic since it is a way to minimize the loss with respect to the decision function.

4.1. Rule extraction via Genetic Algorithm

Searching for the best feature is not an easy task because of the huge dimensionality of the feature space and hence, in general, an exhaustive search is not feasible. In order

to find the best rule, we adopted (as a proof of concept) a genetic algorithm (GA) based optimization. The design choices for the GA are described in the following:

population it is formed by 500 randomly initialized individuals, i.e., mDNF formulas with at most D conjunctions made of at most C literals.

fitness given a formula f, its fitness is equal to the weight w_f as in Eq. (2).

crossover given two mDNF formulas f and g, the crossover operator creates a new individual by randomly selecting a subset of the conjunctive clauses from the union of f and g while keeping the number of clauses $\leq D$.

mutation given a mDNF formula, the mutation operator randomly performs one out of the following three actions:

- removing one of the conjunctive clauses (when applicable);
- adding a new random conjunctive clause;
- replacing a literal in one of the conjunctions with another literal picked from the ones that are not currently included in it.

selection we adopted the *elitist selection* strategy to guarantee that the solution quality will not decrease.

The complete procedure for extracting the best rule is describe in Algorithm 1.

5. Experiments

We experimentally analyzed the proposed approach on different aspects. Specifically:

- fidelity of the extracted interpretation rules with respect to the source SVM on datasets with a specific classification rule;
- benefit of leveraging on the support vectors, via the fitness function, instead of using the genetic algorithm guided by the pure training error;

Algorithm 1: Interpretation rule extraction algorithm

Input:
X: input instances;
y: instances label;
$\kappa_*^{D,C}$: kernel function;
C: conjunction arity;
D: disjunction arity;
P_μ: mutation probability;
N_e: size of the elitist selection;
N_p: maximum number of individuals;
N_g: maximum number of generations;
Output: R: best interpretation rule

$\boldsymbol{\alpha}, SV \leftarrow \text{SVM}(\mathbf{X}, \mathbf{y}, \kappa_*^{D,C})$ ▷ gets the support vectors (SV) and the associated weights ($\boldsymbol{\alpha}$)
$pop \leftarrow random\_initialization(N_p, C, D)$
for $g \leftarrow 1$ **to** N_g **do**
  $pop \leftarrow selection(\boldsymbol{\alpha}, SV, pop, N_e)$ ▷ elitist selection on the basis of the fitness
  do
    $a \leftarrow random\_choice(pop)$
    $b \leftarrow random\_choice(pop)$
    $h \leftarrow mutate(crossover(a, b), P_\mu)$
    $pop \leftarrow pop \cup \{h\}$
  while $|pop| < N_p$
$R \leftarrow$ individual $\in pop$ with maximum fitness
return R

- application on real world categorical datasets;
- adaptation to real-valued datasets.

5.1. Experimental settings

All the experiments have been implemented in python 2.7 using the modules Scikit-Learn [9], MKLpy and pyros available in the PyPi repository.

We evaluated the proposed algorithm in terms of the most used metrics for evaluating explanation rules [2], namely, *comprehensibility*, *accuracy* and *fidelity*. Comprehensibility is the extent to which the extracted representations are humanly comprehensible. In our case we can assume high comprehensibility because the retrieved rules are simple (and short) logical propositions over the input binary variables. The

accuracy of a classification function (or rule) f over the test set $\mathcal{T}_{\text{ts}}$ is equal to

$$accuracy(f, \mathcal{T}_{\text{ts}}) = \frac{|\{(\mathbf{x}, y) \in \mathcal{T}_{\text{ts}} \mid [\![f(\mathbf{x}) \iff y = +1]\!]\}|}{|\mathcal{T}_{\text{ts}}|}.$$

The *fidelity* over the test set $\mathcal{T}_{\text{ts}}$ of a rule f w.r.t. a decision function h learnt by a learning algorithm is computed by

$$fidelity(f, h, \mathcal{T}_{\text{ts}}) = \frac{|\{(\mathbf{x}, y) \in \mathcal{T}_{\text{ts}} \mid [\![f(\mathbf{x}) \iff h(\mathbf{x}) = +1]\!]\}|}{|\mathcal{T}_{\text{ts}}|}.$$

5.2. Case study: toy problems

In this series of experiments we used three UCI [6] datasets (`monks-1`, `monks-3` and `tic-tac-toe`) and seven artificial datasets, for a total of 10 binary datasets. These datasets have a fixed number of ones for every instance. This is not a limitation since, given a dataset with categorical features, each instance can be converted into a fixed norm binary vector by means of the one-hot encoding [10]. The artificial datasets (indicated by the prefix `art-`) have been created in such a way that the positive class can be described by one monotone DNF formula over the input variables. The details of the datasets are summarized in Table 1.

Dataset	#Inst.	#Ft.	Rule
`tic-tac-toe*`	958	27	mDNF[1], $d = 8, c = 3$
`monks-1*`	432	17	$(x_0 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_5) \vee x_{11}$
`monks-3*`	432	17	mDNF[2], $d = 7, c = 2$
`art-d2-c4`	1000	30	$(x_{11} \wedge x_9 \wedge x_1 \wedge x_{14}) \vee (x_{27} \wedge x_{17})$
`art-d3-c3`	1000	30	$(x_3 \wedge x_{28}) \vee x_{27} \vee (x_{14} \wedge x_7 \wedge x_{26})$
`art-d4-c2`	1000	30	$x_3 \vee (x_0 \wedge x_7) \vee (x_5 \wedge x_9) \vee x_8$
`art-d4-c3`	1000	30	$(x_{25} \wedge x_{21}) \vee (x_{15} \wedge x_5 \wedge x_{19}) \vee$ $(x_0 \wedge x_{26}) \vee (x_8 \wedge x_{21} \wedge x_{20})$
`art-d5-c4`	1000	30	mDNF[3], $d = 5, c \leq 4$
`art-d5-c5`	1000	30	mDNF[4], $d = 5, c \leq 5$

Table 1: Information of the datasets: number of instances, number of binary features and the rule which describes the positive class. (*) means that the dataset is freely available in the UCI repository.

This set of experiments aimed to show that the interpretation rules extracted from the SVM are highly faithful to the model.

Dataset	SVM Train	SVM Test	Best Rule Train	Best Rule Test	Fidelity Train	Fidelity Test	GA #Gen.
tic-tac-toe	100.00 ±0.00	98.33 ±0.87	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	98.33 ±0.87	358.00 ±156.81
monks-1	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	9.20 ±3.37
monks-3	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	230.40 ±385.79
art-d2-c4	100.00 ±0.00	98.87 ±0.50	99.89 ±0.23	99.40 ±0.80	99.89 ±0.23	99.07 ±0.68	10.60 ±3.55
art-d3-c3	100.00 ±0.00	97.13 ±1.13	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	97.13 ±1.13	14.60 ±7.34
art-d4-c2	100.00 ±0.00	97.87 ±0.75	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	97.87 ±0.75	15.0 ±2.28
art-d4-c3	100.00 ±0.00	95.07 ±1.34	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	95.07 ±1.34	35.20 ±19.36
art-d5-c4	100.00 ±0.00	96.00 ±0.67	99.71 ±0.57	99.20 ±1.60	99.71 ±0.57	96.00 ±0.67	340.40 ±338.10
art-d5-c5	100.00 ±0.00	94.27 ±0.85	99.97 ±0.06	99.40 ±0.33	99.97 ±0.06	94.20 ±0.85	61.20 ±17.68

Table 2: Experimental results averaged over 5 runs: for each dataset the accuracy (%) in both training and test is reported for SVM and for the extracted rule. It is also reported the fidelity of the rule w.r.t the SVM as well as the average number of generations required to the GA to find the best rule.

For each dataset the experiments have been repeated 5 times by using different 70%-30% training-test splits. In each experiment an hard-SVM with the kernel $\kappa_*^{5,10}$ has been trained over the training set, and then the most relevant formula has been extracted. We used an hard SVM because we know a-priori that all the toy datasets are linearly separable.

The parameters of the GA (described in Section 4.1) have been set as in the following: $C = 5$, $D = 10$, mutation probability = 0.6, and the maximum number of generations = 10^3. It is worth to notice that the computational time for calculating κ_* is in the order of milliseconds for each dataset.

The achieved results are summarized in Table 2.

As evident from the table, in every dataset the best rule extracted by the GA is indeed the one which (almost always) explains the label and the decision of the SVM (the fidelity is very high). Moreover, despite the huge search space (on average 10^{45} for-

[1] $(x_8 \wedge x_{14} \wedge x_{20}) \vee (x_5 \wedge x_{14} \wedge x_{23}) \vee (x_2 \wedge x_{14} \wedge x_{26}) \vee (x_8 \wedge x_{17} \wedge x_{26}) \vee (x_{11} \wedge x_{14} \wedge x_{17}) \vee (x_2 \wedge x_{11} \wedge x_{20}) \vee (x_{20} \wedge x_{23} \wedge x_{26}) \vee (x_2 \wedge x_5 \wedge x_8)$,

[2] $(x_8 \wedge x_{13}) \vee (x_3 \wedge x_{11}) \vee (x_4 \wedge x_{11}) \vee (x_3 \wedge x_{12}) \vee (x_4 \wedge x_{12}) \vee (x_3 \wedge x_{13}) \vee (x_4 \wedge x_{13})$,

[3] $(x_{20} \wedge x_6) \vee (x_3 \wedge x_{17} \wedge x_{16} \wedge x_{23}) \vee (x_1 \wedge x_{27} \wedge x_{12} \wedge x_{29}) \vee (x_{26} \wedge x_{14} \wedge x_{24})$

[4] $(x_{29} \wedge x_8 \wedge x_{15} \wedge x_{14} \wedge x_{19}) \vee (x_{27} \wedge x_9 \wedge x_5) \vee (x_{23} \wedge x_8) \vee (x_{18} \wedge x_{24} \wedge x_9 \wedge x_{11}) \vee (x_{10} \wedge x_1 \wedge x_{12})$

mulas), the number of generations required to find the best rule is very low. Accuracy results are further highlighted in Figure 2.

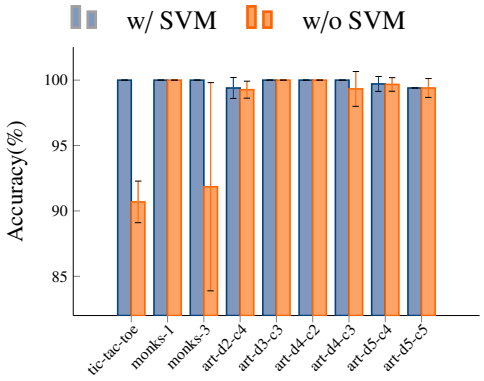


Figure 2: Comparison between the GA guided by the SVM (w/) and w/o the SVM. The plot shows the average accuracy on the test set.

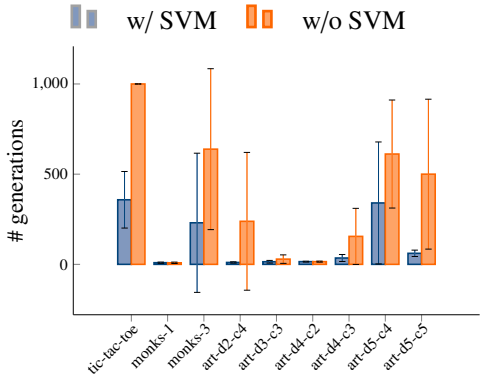


Figure 3: Comparison between the GA guided by the SVM (w/) and w/o the SVM. The plot shows the average number of generations required by the GA to find the best rule.

To highlight how the weights learned by the SVM are indeed useful to guide the research of the GA (through the fitness), we also tried to retrieve the best formula by using the same GA with $\alpha_i = 1/L, \forall\, i \in [1, L]$. In this case the fitness corresponds to the training accuracy. Figure 3 shows the comparison between the GA w/ and w/o SVM. From the figure, it is evident that using the GA guided by the SVM ensures that a better rule will be found with fewer generations. It is also worth to mention that computing the fitness over all the training set is significantly less efficient than

calculating it for the support vectors only. It is worth to notice that the interpretation rules (that we know existing) achieve better accuracies than the associated SVMs. This phenomenon can be explained by the fact that SVM classifies an instance on the basis of a weighted committee of the rules that are satisfied in the support vectors. Since the feature space is composed with a hierarchy of rules, it is reasonable that some of the more general rules (i.e., features) w.r.t. the best one get a not negligible weight. Unfortunately these rules on some instances mislead the classification causing a little drop in the performances.

5.3. Case study: real-world categorical datasets

5.3.1. Poker dataset

The `poker`[5] dataset contains over one million valid poker hands. The general task is to classify the hand. In our experiments we used a subset of the whole dataset by randomly selecting 30 thousand hands. Since we are focusing on binary classification, we selected a target class as the positive one and all the others as negative. This approach is also known as one-vs-rest classification. We tried to classify two different hands, namely, flush and three of a kind. Table 3 show an example of all possible poker hand values and the number of times each of them appear in the dataset.

Hand	Name	Class value	# Instances	%
	Nothing	0	14989	49.96
	Pair	1	12701	42.34
	Two pairs	2	1465	4.88
	Three of a kind	3	606	2.02
	Straight	4	116	0.39
	Flush	5	63	0.21
	Full house	6	44	0.15
	Four of a kind	7	6	0.02
	Straight flush	8	5	0.017
	Royal flush	9	5	0.017

Table 3: Example of all possible poker hand values.

[5] `https://www.kaggle.com/c/poker-rule-induction`

Each poker hand has been converted into a binary vector using a one-hot encoding of each card appended one after the other. A card is one-hot encoded in the following way. Given a card C, with suit $suit(C) \in \{\heartsuit, \diamondsuit, \spadesuit, \clubsuit\}$ and value $val(C) \in \{2, 3, \ldots, 10, \mathrm{A}, \mathrm{J}, \mathrm{Q}, \mathrm{K}\}$, both suit and value are one-hot encoded and finally appended together. For example, the $10\spadesuit$ is converted into a vector $\mathbf{x}_{10\spadesuit} \in \{0, 1\}^{17}$ as:

$$
\begin{array}{r|cccccccccccccc|cccc|c}
 & \mathrm{A} & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \mathrm{J} & \mathrm{Q} & \mathrm{K} & \heartsuit & \diamondsuit & \spadesuit & \clubsuit & \\
x_{10\spadesuit} = [& 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 &]
\end{array}
$$

At the end of this conversion, a poker hand will have a dimension of $17 \times 5 = 85$.

Conversely to the previous set of experiments, here we employed a soft SVM using a validation procedure (5-fold cross validation) in order to choose the best trade-off hyper-parameter C from the set $\{10^{-1}, \ldots, 10^{5}\}$. The kernel has been fixed to $\kappa_*^{5,10}$. 5 runs of a 70%-30% training and test split has been performed. The average results are shown in Table 4. The discussion about the interpretation rules is based on an example of rule extracted in one of the run.

In the following we discuss the two tasks separately in order to highlight the strengths and weaknesses of the extracted interpretation rules.

Flush. In the flush classification task, the interpretation rule extracted by the algorithm is a mDNF(4,5), namely,

$$
\begin{aligned}
&(suit(C_1) = \heartsuit \wedge suit(C_2) = \heartsuit \wedge suit(C_3) = \heartsuit \wedge suit(C_4) = \heartsuit \wedge suit(C_5) = \heartsuit) \vee \\
&(suit(C_1) = \diamondsuit \wedge suit(C_2) = \diamondsuit \wedge suit(C_3) = \diamondsuit \wedge suit(C_4) = \diamondsuit \wedge suit(C_5) = \diamondsuit) \vee \\
&(suit(C_1) = \clubsuit \wedge suit(C_2) = \clubsuit \wedge suit(C_3) = \clubsuit \wedge suit(C_4) = \clubsuit \wedge suit(C_5) = \clubsuit) \vee \\
&(suit(C_1) = \spadesuit \wedge suit(C_2) = \spadesuit \wedge suit(C_3) = \spadesuit \wedge suit(C_4) = \spadesuit \wedge suit(C_5) = \spadesuit) \vee
\end{aligned}
$$

where C_i indicates the i-th card in the hand for some enumeration of the cards. Even though the rule is correct with respect to the concept of flush, the classification accuracy

Target hand	Rule Accuracy	SVM Accuracy	Fidelity
Flush	99.98	99.83	99.81
Three of a kind	93.40	97.80	94.26

Table 4: Results on the `poker` dataset.

is not 100%. This is because all combinations that contain a flush, e.g., royal flush, are not labelled as a simple flush which cause a miss-classification.

Three of a kind. From a human perspective, recognizing a flush, i.e., five cards with the same suit, is roughly as easy as recognizing, for example, a three of a kind. However, expressing such concepts using logical propositions over the values of the single cards is not always trivial. Specifically, a three of a kind (TOAK) can only be expressed by enumerating all possible combinations of values such that three of them are equal. For a poker hand with 5 cards, a logical formula able to discriminate a TOAK is a mDNF(130,3) which is not included in the feature space of $\kappa_*^{5,10}$.

The accuracy achieved in the TOAK task, reported in Table 4, is highly influenced by the unbalanced nature of the dataset. In fact, if we analyze the extracted interpretation rule it is evident that the feature space is not rich enough to mimic the TOAK concept. The following formula is an example of the extracted best rule:

$$(rank(C_1) = \mathrm{A} \wedge rank(C_5) = \mathrm{A}) \vee (rank(C_1) = 7 \wedge rank(C_2) = 7) \vee$$

$$(rank(C_2) = 2 \wedge rank(C_3) = 2) \vee (rank(C_1) = 5 \wedge rank(C_3) = 5) \vee$$

$$(rank(C_1) = 8 \wedge rank(C_2) = 8) \vee (rank(C_1) = 8 \wedge rank(C_5) = \mathrm{Q} \wedge suit(C_2) = \blacklozenge) \vee$$

$$(rank(C_1) = \mathrm{K} \wedge rank(C_3) = \mathrm{K}) \vee (rank(C_4) = 6 \wedge rank(C_5) = 6) \vee$$

$$(rank(C_2) = 4 \wedge rank(C_4) = 4) \vee (suit(C_2) = \clubsuit \wedge rank(C_4) = 8).$$

It is clear that the rule is far from being an explanation rule for a TOAK. It contains some notion of pair but it is not expressive enough to capture all the possibilities of having a TOAK. Moreover, there are also a couple of conjunctive clauses completely irrelevant for a TOAK, namely, $(suit(C_2) = \clubsuit \wedge rank(C_4) = 8)$ and $(rank(C_1) = 8 \wedge rank(C_5) = \mathrm{Q} \wedge suit(C_2) = \blacklozenge)$.

This example shows the limitation of using a logic representation defined over the values of the attributes. In order of being more expressive, there is the need of introducing higher level features that capture, for example, the relation between the attributes.

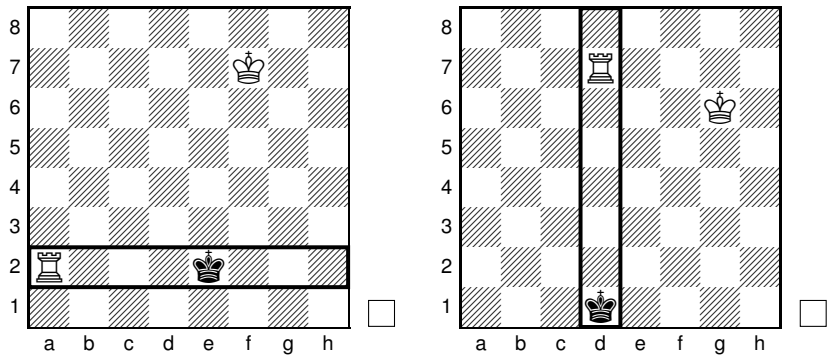
5.3.2. Chess dataset

The `chess`[6] dataset contains a set of 10000 (White) King+Rook vs. (Black) King ending positions. The task is to classify illegal positions w.r.t. the legal ones with

white to move. In this particular case, an illegal position can be due to at least one of the following reasons:

1. The Black King is in check (directly attacked) by the White Rook. This can be achieved by having both White Rook and Black King in the same rank/file, and the White King must not interfere. Figure 4 gives an example of the just mentioned situation;
2. The current position on the board is impossible. E.g., two pieces in the same square, or the two Kings in adjacent squares (mutual check which is impossible to achieve).

The dataset contains 67% of illegal positions against the remaining 33% of legal ones.



(a) White Rook (♖) and Black King (♚) in the same rank

(b) White Rook (♖) and Black King (♚) in the same file

Figure 4: Examples of check: White Rook and Black are aligned. It is white to move, so the position is illegal.

A chess position is converted into a binary vector via the one-hot encoding of each piece square (both the file and the rank are encoded) which are then appended together by keeping the same pieces order. Thus, a position is finally represented as a binary vector of dimension $(8 \times 2) \times 3 = 48$. Also in this case a 5 runs of 70%-30% training and test split has been performed.

It is easy to grasp that, as in the case of the TOAK for the poker, also for this task

[6] `https://www.doc.ic.ac.uk/~shm/Datasets/chess/`

the feature space is not rich enough to contain a rule able to describe an illegal position. We can also observe that there are some analogies between the two tasks. In the TOAK task it is difficult to express the notion of *3 cards with the same rank* because the logical propositions are defined on single values rather than relations between the variables.

Similarly, in the illegal position task, it would be much easier to express such concept having the possibility of using relations between the position of the pieces, for example, the notion *White Rook and Black King have the same rank*.

For this reason, we also tried a second experiment in which we added to each instance 6 features that indicate whether two pieces have the same file/rank. For example, the position in Figure 5 is represented by a binary vector $\mathbf{x} \in \{0, 1\}^{54}$ in which the feature file(♚)=file(♖) is set to 1, as well as the feature rank(♔)=rank(♚).

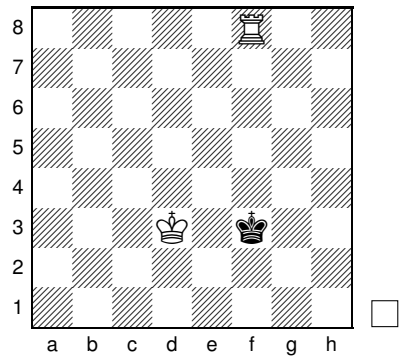


Figure 5: Examples of a possible position.

In the reminder we will refer to this representation as *improved* representation, while the previous one as the *standard* representation. The achieved results using both of the representations are reported in Table 5

Repr	Rule Accuracy	SVM Accuracy	Fidelity
Standard	82.60	96.00	82.80
Improved	90.30	96.00	92.50

Table 5: Results on the `chess` dataset.

It is self-evident that using a slightly improved representation can hugely affect

the quality of the extracted interpretation rule. An example of rule extracted using the standard representation is the following:
(file(♖)=d ∧ file(♚)=d) ∨ (file(♔)=h ∧ file(♚)=h) ∨ (rank(♖)=2 ∧ rank(♚)=2) ∨ (file(♖)=h ∧ file(♚)=h) ∨(file(♖)=g ∧ file(♚)=g) ∨ (rank(♖)=7 ∧ rank(♚)=7) ∨ (rank(♖)=1 ∧ rank(♚)=1) ∨ (rank(♖)=8 ∧ rank(♚)=8) ∨ (rank(♖)=3 ∧ rank(♚)=3) ∨ (file(♖)=e ∧ file(♚)=e).

The rule catches some of the possible combinations of files and ranks which can cause an illegal position, however they are not nearly enough. Let us see instead a rule extracted in the improved representation:
(rank(♔)=5 ∧ rank(♖)=6 ∧ file(♚)=file(♔)) ∨ (file(♔)=d ∧ file(♚)=e) ∨ (file(♔)=d ∧ file(♚)=c) ∨ (file(♔)=b ∧ file(♚)=c) ∨ (file(♔)=b ∧ file(♚)=a) ∨ (rank(♔)=5 ∧ rank(♚)=4 ∧ file(♔)=file(♚))∨ (file(♚)=file(♔) ∧ rank(♚)=rank(♔)) ∨ (file(♖)=file(♔) ∧ rank(♖)=rank(♔)) ∨ (file(♖)=file(♚)) ∨ (rank(♖)=rank(♚))

Despite not being entirely correct, the rule covers many cases of illegal position. For example, the last four conjunctions almost explain all the possible ways of making an illegal position: ♔and ♚ in the same square, ♔ and ♖ in the same square, ♚ and ♖ in the same file, and ♚ and ♖ in the same rank.

This experiment underline the need of richer representations especially higher level features that express concept between the variables values which are missing in the common Boolean kernels feature spaces.

5.4. Case study: real world real-valued dataset

As pointed out previously, one of the shortcomings of Boolean kernels is the unapplicability to not binary valued data. However, it is possible to overcome this problem using a discretization method. Discretization is a preprocessing technique which aims to transform a set of continuous attributes into discrete ones, by associating categorical values to intervals and thus transforming quantitative data into qualitative data. Clearly, discretizing the data can strongly affect the effectiveness of the SVM which tightly depends on the applied discretization technique. Hence, there is the need of making the right choice of the discretization method.

5.4.1. MDLP [11]

The binary discretization approach presented in [11] is a generalization of the standard entropy-based method used in decision trees [12, 13, 14]. The algorithm decides to partition an interval using a criterion based on the minimum description length principle: the partition induced by a cut point is accepted if and only if the length of the message required to send before partition is more than the length of the message require to send after partition [15]. In a recent survey [16] the MDLP method has been considered by the authors as one of the best alternatives in terms of tradeoff between the number of intervals produced and accuracy.

5.4.2. Breast-cancer dataset

The Wisconsin Breast Cancer dataset[7] (`wbcd`) is a standard UCI [6] dataset which contains hospital patients values captured via a FNA (Fine-needle aspiration) test. Each patient is characterized by 9 attributes regarding breast tumoral cells as described in Table 6. Patients with missing values have been removed.

Domain	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Clump Thickness (CT)	139	50	104	79	128	33	23	44	14	69
Uniformity of cell size (UCSI)	373	45	52	38	30	25	19	28	6	67
Uniformity of cell shape (UCSH)	346	58	53	43	32	29	30	27	7	58
Marginal adhesion (MA)	393	58	58	33	23	21	13	25	4	55
Single Epithelial cell size (SECS)	44	376	71	48	39	40	11	21	2	31
Bare Nuclei (BN)	402	30	28	19	30	4	8	21	9	132
Bare Chromatin (BC)	150	160	161	39	34	9	71	28	11	20
Normal Nucleoli (NN)	432	36	42	18	19	22	16	23	15	60
Mitoses (M)	563	35	33	12	6	3	9	8	0	14

Table 6: `wbcd` dataset description. Each attribute can assume an integer value in the range [1,10].

The task consists in correctly classify malignant breast tumor. Before applying our algorithm the dataset has been discretized using the MDLP method describe previously. This dataset gives us the chance to compare the extracted interpretation rule with other rule extraction methods. In particular we use as reference the results reported in [17]. Since in [17] experiments have been carried out using a 10-fold cross

[7] `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`

Method	#Rules	Rules	Acc.(%)
SSV[18]	3	MA> 2.5 $\wedge$ BC> 2.5 MA> 2.5 $\wedge$ BN> 3.5 $\wedge$ BC> 0 UCSI> 5.5 $\wedge$ MA< 2.5 $\wedge$ BC> 1.6	86.36
NEFC.[19]	1	CT is large $\wedge$ UCSH is large $\wedge$ MA is large $\wedge$ BN is large $\wedge$ BC is large $\wedge$ NN is large	82.26
GASVM[20]	2	CT$\geq$ 7.085 $\vee$ UCSH $\geq$ 7.908 $\vee$ SECS $\geq$ 9.76 $\vee$ BC$\geq$ 6.064 UCSH< 7.7 $\wedge$ BN< 9.41 $\wedge$ BC< 6.12 $\wedge$ M< 7.43	92.38
QSVM-G[21]	7	UCSH> 2.97063 $\wedge$ BN> 4.93843 CT> 4.96292 $\wedge$ UCSI> 3.9883 UCSI> 4.97866 CT> 2.98416 $\wedge$ BN> 6.92955 CT> 5.98057 $\wedge$ UCSI> 2.99794 $\wedge$ UCSH> 3.98945 UCSH> 2.97063 $\wedge$ SECS> 4.96611 NN> 8.964	97.36
ReRXJ48[22, 17]	2	CT $\leq$ 4 $\wedge$ BN> 6 CT> 4 $\wedge$ BN> 1	90.47
Our[23]	1	CT> 4.5 $\vee$ UCSI> 2.5 $\vee$ 2.5 <UCSH $\leq$ 4.5	89.73

Table 7: Rule extracted by some of the rule extraction algorithms reported in [17]. The accuracy referred to the performance of the set of rules (appended with an 'or') on the whole `wbcd` dataset.(*) this result has been achieved by substituting 'is large' with greater than the average value of the attribute.

validation, we used the same procedure. In this way the size of the test set is preserved. The comparison is done by applying the extracted rule/rules to the entire dataset, and the overall accuracy is registered. Table 7 summarizes the results. In the table only the best performing methods w.r.t. [17] have been reported.

The first noticeable observation is that, in general, rule extraction methods return a set of rules w.r.t. the single rule provided by our algorithm. Another difference are the disjunctive clauses: in a standard rule extraction method they are inserted a-posteriori in order to put all the rules together into a single one. Our method instead, by design, gives a single mDNF which can contain disjunctions. From an accuracy point of view, the rule extracted by our algorithm achieve a performance which is comparable to most of the rule sets. The only method which achieve very high accuracy is QSVM-G, but at a cost of a quite complex rule.

Another observation is that the rules of the best performing methods, namely, QSVM-G, GASVM, ReRXJ48 and our, have rules that include almost the same at-

tributes. Especially, from the results it seems that large values of CT and UCSH are a good indicator of malignant breast tumor.

It is worth to underline that, the compared methods have been design to extract rules able to explain the dataset. In turn, our proposed method works on top of a BK-SVM and it extracts rule to interpret the solution of such SVM.

6. Future work

We have presented a method based on Boolean kernels to extract interpretation rules from a support vector machine. After showing the effectiveness on categorical datasets, we also provide a method for applying the proposed algorithm to real-valued data. We have also investigated the strengths and weaknesses of our proposal. In particular, it is evident that in many real-world problems using simple propositions over the value of the features is not always enough. A richer representation is needed in order to give the possibility to the SVM of leveraging on such higher level features. The `chess` dataset has been used to show how effective can be the addition of higher level features. To this end, in the future we aim to build methods able to learn such higher level representations. Moreover, a more in-depth theoretical analysis need to be conducted in order to further support the empirical results. Finally, other strategies for extracting the best rules should be tested.

References

[1] Bryce Goodman and Seth Flaxman. Eu regulations on algorithmic decision-making and a "right to explanation", 2016. presented at 2016 ICML WHI 2016, New York, NY.

[2] Nahla Barakat and Andrew P. Bradley. Rule extraction from support vector machines: A review. *Neurocomputing*, 74(1-3):178–190, 2010.

[3] João Guerreiro and Duarte Trigueiros. A unified approach to the extraction of rules from artificial neural networks and support vector machines. In *ADMA*, pages 34–42, 2010.

[4] Mirko Polato, Ivano Lauriola, and Fabio Aiolli. Classification of categorical data in the feature space of monotone dnfs. In *ICANN*, pages 279–286, 2017.

[5] Mirko Polato, Ivano Lauriola, and Fabio Aiolli. A novel boolean kernels family for categorical data. *Entropy*, 20(6), 2018.

[6] M. Lichman. UCI machine learning repository, 2013.

[7] Ken Sadohara. Learning of boolean functions using support vector machines. In *ALT*, pages 106–118, 2001.

[8] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, Boston, 2013.

[11] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029, 1993.

[12] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[13] J. R. Quinlan. Machine learning. chapter Probabilistic Decision Trees, pages 140–152. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[14] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[15] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, 2002.

[16] S. García, J. Luengo, J. A. Sáez, V. López, and F. Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[17] Yoichi Hayashi and Satoshi Nakano. Use of a recursive-rule extraction algorithm with j48graft to achieve highly accurate and concise rule extraction from a large breast cancer dataset. *Informatics in Medicine Unlocked*, 1:9 – 16, 2015.

[18] Detlef Nauck and Rudolf Kruse. Obtaining interpretable fuzzy classification rules from medical data. *Artificial Intelligence in Medicine*, 16(2):149 – 169, 1999.

[19] W. Duch, R. Adamczak, and K. Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions on Neural Networks*, 12(2):277–306, 2001.

[20] Yan-Cheng Chen, Chao-Ton Su, and Taho Yang. Rule extraction from support vector machines by genetic algorithms. *Neural Computing and Applications*, 23(3):729–739, 2013.

[21] Guido Bologna and Yoichi Hayashi. Qsvm: A support vector machine for rule extraction. In *Advances in Computational Intelligence*, pages 276–289, Cham, 2015. Springer International Publishing.

[22] Yoichi Hayashi, Yuki Tanaka, Tomohiro Takagi, Takamichi Saito, Hideaki Iiduka, Hiroaki Kikuchi, Guido Bologna, and Sushmita Mitra. Recursive-rule extraction algorithm with j48graft and applications to generating credit scores. *J. Artif. Intell. Soft Comput. Res.*, 6:35, 2016.

[23] Mirko Polato, Ivano Lauriola, and Fabio Aiolli. Boolean kernels for interpretable kernel machines. In *ESANN*, 2018.