entropy

MDPI

Article

A Novel Boolean Kernels Family for Categorical Data †

Mirko Polato [1,*], Ivano Lauriola [1,2] and Fabio Aiolli [1]

[1] Department of Mathematics, University of Padova, via Trieste 63, 35121 Padova, Italy;
[2] Fondazione Bruno Kessler, via Sommarive 18, 38123 Trento, Italy; ivano.lauriola@phd.unipd.it (I.L.); aiolli@math.unipd.it (F.A.)
* Correspondence: mpolato@math.unipd.it
† This paper is an extended version of our paper published in the 26th International Conference on Artificial Neural Networks—ICANN 2017.

Received: 28 February 2018; Accepted: 4 June 2018; Published: 6 June 2018

Abstract: Kernel based classifiers, such as SVM, are considered state-of-the-art algorithms and are widely used on many classification tasks. However, this kind of methods are hardly interpretable and for this reason they are often considered as *black-box* models. In this paper, we propose a new family of Boolean kernels for categorical data where features correspond to propositional formulas applied to the input variables. The idea is to create human-readable features to ease the extraction of interpretation rules directly from the embedding space. Experiments on artificial and benchmark datasets show the effectiveness of the proposed family of kernels with respect to established ones, such as RBF, in terms of classification accuracy.

Keywords: boolean kernels; kernel methods; SVM; categorical data

1. Introduction

Large-margin kernel machines (e.g., SVM) are recognized state-of-the-art algorithms in machine learning applications. They are broadly applied to several domains, such as text categorization, spam filtering, RNA function prediction, and so on. However, since these methods typically work on an implicitly defined feature space by resorting to the well-known kernel trick, the interpretability of the resulting model is difficult.

This last aspect is often crucial in specific application areas, such as the medical ones, in which the simple predictive answer is not enough. Being this a requirement for the acceptance of these *black-box* models by end users, in the last decade, several methods have been introduced for rule extraction from SVMs (see [1] for a recent survey). The majority of the proposed approaches try to extract *if–then* rules over the input variables and this task is generally hard when common kernels, e.g., the polynomial and RBF ones, are used.

On the other hand, Decision Trees (DT), thanks to their easy logical interpretation, are very appreciated, especially by non-expert users. The shortcoming of DTs is that, in general, they are not as accurate as more complex methods. In the case of binary valued input data, an alternative approach to make SVM more interpretable consists in defining features that are easy to interpret, for example, features that are propositional (i.e., logical) formulas applied to the input vectors. In particular, Boolean kernels are kernel functions in which the binary input vectors are mapped into an embedding space formed by propositional formulas of the input variables, and, in such space, the dot product is performed.

More formally, a Boolean kernel function $\kappa : \{0,1\}^n \times \{0,1\}^n \to \mathbb{N}$ is defined as $\kappa(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$ where $\phi : \{0,1\}^n \to \{0,1\}^N$ is the embedding Boolean function with $\boldsymbol{x}, \boldsymbol{z} \in \{0,1\}^n$. When the input space is binary, the linear kernel can be seen as a limit case of a Boolean kernel in which the features simply represent the Boolean literals themselves.

In this paper, we propose a new family of Boolean kernels able to produce feature spaces composed by arbitrarily complex propositional formulas. In particular, we first introduce the basic Boolean kernels [2], namely the conjunctive kernel and the disjunctive kernel, for both the monotone and non-monotone case. On top of these kernels, we then propose more complex kernels such as the Disjunctive Normal Form kernel and the Conjunctive Normal Form kernel. For all the proposed kernels, an efficient method to compute them is provided. We assess the quality of the proposed kernels in terms of classification accuracy on several categorical datasets, and compare their performance against state-of-the-art kernels, such as the RBF kernel, and other Boolean kernels proposed in the literature.

The reminder of this paper is structured as follows: in Section 2, we give an overview of the existing work related to Boolean kernels. In Section 3, we present the proposed Boolean kernels family and, in Section 4, we discuss their computational complexity. Finally, Section 5 shows all the performed experiments on several benchmark categorical datasets.

2. Related Work

Sadohara [3] was the first to introduce the idea of Boolean kernel. In that work, the concept of Boolean kernel is actually related to a single kernel called DNF kernel. Specifically, he proposed a SVM for learning Boolean functions: since every Boolean (i.e., logical) function can be expressed in terms of Disjunctive Normal Form (DNF) formulas, the proposed kernel creates a feature space containing all possible conjunctions of negated or non-negated Boolean variables.

For instance, the feature space for a two variables, e.g., x_1, x_2, DNF contains the following $3^2 - 1$ features: $x_1, x_2, \neg x_1, \neg x_2, x_1 \wedge x_2, x_1 \wedge \neg x_2, \neg x_1 \wedge x_2, \neg x_1 \wedge \neg x_2$. The resulting decision function of a kernel machine which employs the DNF kernel can be represented as a weighted linear sum of conjunctions (Representer Theorem [4,5]), which in turn can be seen as a kind of "soft" DNF.

Formally, the DNF kernel between $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^n$ is defined as

$$\kappa_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + \prod_{i=1}^{n} (2\boldsymbol{x}_i \boldsymbol{z}_i - \boldsymbol{x}_i - \boldsymbol{z}_i + 2),$$

while its monotone (i.e., without negations) form is the following

$$\kappa_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + \prod_{i=1}^{n} (\boldsymbol{x}_i \boldsymbol{z}_i + 1).$$

By restricting the domain of the vectors in $\{0,1\}^n$, the computation of the kernels is simplified as follows

$$\kappa_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + 2^{\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \overline{\boldsymbol{x}}, \overline{\boldsymbol{z}} \rangle}, \quad \kappa_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + 2^{\langle \boldsymbol{x}, \boldsymbol{z} \rangle},$$

where $\overline{\boldsymbol{x}} = \mathbf{1}_n - \boldsymbol{x}$ is the vector with all the binary entries swapped. Note that the sum $\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \overline{\boldsymbol{x}}, \overline{\boldsymbol{z}} \rangle$ simply counts the number of common "bits" between $\boldsymbol{x}$ and $\boldsymbol{z}$. These last two kernels were independently discovered in [6,7].

A drawback of this type of kernels is the exponential growth of the size of the feature space with respect to the number of involved variables, i.e., $3^n - 1$ for n variables. To give the possibility of controlling the size of the feature space, Sadohara et al. [8] proposed a variation of the DNF kernel in which only conjunctions with up to d variables (i.e., d-ary conjunctions) are considered. Over binary vectors, this kernel, dubbed d-DNF kernel, is defined as

$$\kappa^d_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{i=1}^{d} \binom{\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \overline{\boldsymbol{x}}, \overline{\boldsymbol{z}} \rangle}{i},$$

and trivially, if $d = n$, $\kappa^d_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = \kappa_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z})$. A nice property of the d-DNF kernel is that it yields a nested sequence of hypothesis spaces, i.e., $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_n$. Thus, choosing a degree d (also known as "arity") for the kernel implicitly means controlling the capacity of the hypothesis space,

which is a very important aspect in learning. The same "trick" can be applied to the monotone d-DNF kernel [9]:

$$\kappa^d_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{i=1}^{d} \binom{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{i}.$$

Instead of limiting the number of involved variables, Zhang et al. [10] proposed a parametric version of the DNF and mDNF kernel. Specifically, given $\boldsymbol{x}, \boldsymbol{z} \in \{0,1\}^n$ and $\sigma > 0$, then

$$\kappa^{(\sigma)}_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + \prod_{i=1}^{n} (\sigma \boldsymbol{x}_i \boldsymbol{z}_i + \sigma(1 - \boldsymbol{x}_i) + \sigma(1 - \boldsymbol{z}_i) + 1),$$
$$\kappa^{(\sigma)}_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) = -1 + \prod_{i=1}^{n} (\sigma \boldsymbol{x}_i \boldsymbol{z}_i + 1).$$

The parameter σ induces an inductive bias towards simpler or more complex DNF formulas. Specifically, for σ in the range $[0,1]$ a bias towards shorter DNF is given, while for $\sigma > 1$ the bias is more towards longer DNF. When $\sigma = 1$, then $\kappa^{(\sigma)}_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) = \kappa_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z})$, and the same for the monotone DNF kernel. In the following, we refer to these kernel as σ-DNF and σ-mDNF kernel, respectively. Zhang et al. also proved that, for binary input vectors, the polynomial kernel, i.e., $\kappa^p_{\text{POLY}}(\boldsymbol{x}, \boldsymbol{z}) = (\sigma \langle \boldsymbol{x}, \boldsymbol{z} \rangle + c)^p$, $c \in \mathbb{R}_+$, is a Boolean kernel, even though they did not provide any formal definition of Boolean kernel. Nonetheless, an important observation is that the embedding space of a polynomial kernel is composed by all the monomials (that are conjunctions) up to the degree p. Thus, the only difference between the polynomial one and the d-DNF kernel are the weights associated to the features. It is also noteworthy that, in the binary case, the embedding of the polynomial kernel contains sets of equivalent features, e.g., for $p = 4$ and $\boldsymbol{x} \in \{0,1\}^2$, the value of the features (in the feature space) $\boldsymbol{x}_1^3 \boldsymbol{x}_2, \boldsymbol{x}_1^2 \boldsymbol{x}_2^2, \boldsymbol{x}_1 \boldsymbol{x}_2^3$ are equivalent to the feature $\boldsymbol{x}_1 \boldsymbol{x}_2$.

A kernel related to the polynomial is the all-subset kernel [11,12], defined as

$$\kappa_{\subseteq}(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{n} (\boldsymbol{x}_i \boldsymbol{z}_i + 1),$$

which considers a space with a feature for each subset of the input variables, including the empty subset. It is different from the polynomial because it does not limit the number of considered monomials, and it gives the same weight to all the features. It is easy to see that the all-subset kernel and the monotone DNF kernel are actually the same kernel up to the constant -1, i.e., $\kappa_{\subseteq}(\boldsymbol{x}, \boldsymbol{z}) = \kappa_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) + 1$.

Both the polynomial and the all-subsets kernel have limited control of which features they use and how they are weighted. The polynomial kernel uses only monomials of degree up to p with a weighting scheme depending on a parameter (c). The all-subsets, instead, makes use of the monomials corresponding to all possible subsets of the n input variables.

A restricted version of the all-subset kernel is the ANOVA kernel [11] in which the embedding space is formed by monomials with a fixed degree d without repetition. For example, given $\boldsymbol{x} \in \{0,1\}^3$ the feature space of the all-subset kernel would be made by the features $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_1 \boldsymbol{x}_2, \boldsymbol{x}_1 \boldsymbol{x}_3, \boldsymbol{x}_2 \boldsymbol{x}_3, \boldsymbol{x}_1 \boldsymbol{x}_2 \boldsymbol{x}_3$ and $\emptyset$, while for the ANOVA kernel of degree 2 it would be composed by $\boldsymbol{x}_1 \boldsymbol{x}_2, \boldsymbol{x}_1 \boldsymbol{x}_3$ and $\boldsymbol{x}_2 \boldsymbol{x}_3$. Formally, the ANOVA kernel is defined as follows

$$\kappa^d_{\text{A}}(\boldsymbol{x}, \boldsymbol{z}) = \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} \prod_{j=1}^{d} \boldsymbol{x}_{i_j} \boldsymbol{z}_{i_j},$$

where $i_1, i_2, \ldots, i_d$ are all the possible sets of indices of cardinality d, taken from the set $\{1, \ldots, n\}$.

In [13,14], Boolean kernels are used for studying the learnability of logical formulas, specifically DNF formulas, using maximum margin algorithms, such as SVM. In particular, in [14], the authors showed the learning limitations of some Boolean kernels inside the PAC (Probably Approximately

Correct) framework. Moreover, in [13], Kowalczyk at al. proposed a generalization of the Sadohara mDNF kernel. A special case of this kernel is represented by the σ-mDNF kernel.

From the application point of view, Boolean kernels have been successfully applied on many learning tasks, such as face recognition [15,16], spam filtering [17], load forecasting [18], and generic binary classification tasks [8,10].

3. A New Family of Boolean Kernels

In this section, we propose a new family of Boolean kernels which owns the characteristic of creating feature spaces that are very easy to understand, since they are based on logic. Specifically, features are logical formulas (of a fixed form) over the input Boolean variables.

Firstly, we present the most basic Boolean kernel and then, for both the *monotone* and the *non-monotone* cases, we propose kernels which mimic the conjunctive operator (*and*) and the disjunctive operator (*or*). Then, we provide an efficient way to combine these "base" Boolean kernels to obtain more complex ones, such as kernels with feature spaces composed by normal form formulas.

Throughout the paper, unless specified otherwise, we refer to vectors $\boldsymbol{x}, \boldsymbol{z} \in \{0,1\}^n$ as binary (Boolean) vectors of dimension $n \in \mathbb{N}_+$. We also use $\mathcal{X} \equiv \{i \,|\, x_i = 1\}$ and $\mathcal{Z} \equiv \{i \,|\, z_i = 1\}$ as the sets interpretation of those vectors, while the set $\mathcal{U} \equiv \{1, \ldots, n\}$ indicates the universal set. Given a set $\mathcal{A}$, we refer to its i-th element with $\mathcal{A}_i$ for some enumeration of the elements of $\mathcal{A}$. With the notation $[\mathcal{A}]^k \equiv \{\mathcal{S} \subseteq \mathcal{A} \mid |\mathcal{S}| = k\}$, we express the set of all the subsets of $\mathcal{A}$ of cardinality k. It is worth noticing that, for any binary vector $\boldsymbol{x}$, $\|\boldsymbol{x}\|_2^2 = \|\boldsymbol{x}\|_1$ holds, which is the number of ones contained in it. For the sake of brevity, we refer to this quantity with $|\boldsymbol{x}|$. Moreover, $\mathbf{1}_n$ denotes the n-dimensional vector with all entries equal to 1, and with the notation $\langle \cdot, \cdot \rangle$ we refer to the dot product. The symbol $\odot$ denotes the entry-wise multiplication between matrices. Finally, given a function $f : \mathcal{X} \to \mathcal{Y}^n$, $\mathcal{Y} \subseteq \mathbb{R}$, then $|f|$ denotes the dimension of its codomain, i.e., n.

For each of the proposed kernels, the embedding function is provided in the general form $\phi : \boldsymbol{x} \mapsto \phi^{(b)}(\boldsymbol{x})_{b \in \mathbb{B}}$, where $\mathbb{B}$ is a set of Boolean functions (formulas) over variables of $\boldsymbol{x}$ such that $\phi^{(b)}(\boldsymbol{x}) = b(\boldsymbol{x})$ is a truth value associated with the application of the formula b to $\boldsymbol{x}$. For example, let $b(\boldsymbol{x}) = x_1 \wedge x_2$ and $\mathbf{v} = [0,1,0,1]$, then $b(\mathbf{v}) = 0$, that is *false*. For the sake of simplicity, for each new kernel, only the set $\mathbb{B}$ from which the Boolean formulas are taken is defined.

3.1. Monotone Boolean Kernels

A Boolean function (or formula) $f : \{0,1\}^n \to \{0,1\}$ is called *monotone* if replacing a 0 (i.e., *false*) with a 1 (i.e., *true*) in the input can only increase f's value, i.e., the truth value can only change from *false* to *true*. In other words, a formula f is *monotone* if it does not have any *not* operator.

3.1.1. Monotone Literal Kernel

In logic, a literal is an atomic formula or its negation. Here, we are in the monotone setting, so we refer to a literal only in its positive form. In this case, the embedding is formed by Boolean literals taken from the set $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = x_i\}$. Hence, the monotone Literal (mL) kernel, $\kappa_{\text{mL}}(\boldsymbol{x}, \boldsymbol{z})$, will count how many *true* (i.e., positive) input literals the vectors have in common. Actually, κ_{mL} is exactly the linear kernel $\kappa_{\text{LIN}}(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle$, which simply performs the dot product between the two input binary vectors.

3.1.2. Monotone Conjunctive Kernel

In Boolean algebra, given two Boolean variables $x, z \in \{0,1\}$, the conjunction (i.e., *and*) between x and z, denoted by $x \wedge z$, is satisfied if and only if $x = z = 1$, that is if and only if both variables are *true*. Given two vectors $\boldsymbol{x}, \boldsymbol{z} \in \{0,1\}^n$, the monotone Conjunctive (mC) kernel [19] $\kappa_{\text{mC}}^c(\boldsymbol{x}, \boldsymbol{z})$ counts how many monotone conjunctions of the input variables, of a fixed arity c, are satisfied in both $\boldsymbol{x}$ and $\boldsymbol{z}$. In particular, the embedding is defined by Boolean formulas taken from $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = \bigwedge_{j \in [\mathcal{U}]_i^c} x_j\}$, which represent all the possible conjunctions of c literals (i.e., variables) taken from $\boldsymbol{x}$. The dimension

of the resulting feature space is $\binom{n}{c}$, that is the number of all the combinations of c different variables taken from the input n-dimensional space. To count all the possible conjunctions of c variables satisfied in both $\boldsymbol{x}$ and $\boldsymbol{z}$, we need to calculate the number of combinations of c monomials that can be formed by using all the positive variables in both the vectors, that is the value of the kernel $\kappa_{\text{mL}}(\boldsymbol{x}, \boldsymbol{z})$. Hence, we obtain:

$$\kappa^{c}_{\text{mC}}(\boldsymbol{x}, \boldsymbol{z}) = \binom{\kappa_{\text{mL}}(\boldsymbol{x}, \boldsymbol{z})}{c} = \binom{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{c}.$$

It is easy to see that for binary vectors κ^{c}_{mC} is actually the ANOVA kernel of degree c [11].

As shown in [19], we can express the Sadohara mDNF Kernel [3] as a linear combination of mC-kernels of arity $1 \leq d \leq n$ as in the following:

$$\kappa_{\text{mdnf}}(\boldsymbol{x}, \boldsymbol{z}) = 2^{\langle \boldsymbol{x}, \boldsymbol{z} \rangle} - 1 = \sum_{d=0}^{n} \binom{\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{d} - 1 = \sum_{d=1}^{n} \kappa^{d}_{\text{mC}}(\boldsymbol{x}, \boldsymbol{z}).$$

A similar construction also holds for the d-mDNF kernel.

3.1.3. Monotone Disjunctive Kernel

The disjunction of two Boolean variables $x, z \in \{0, 1\}$, denoted by $x \vee z$, is not satisfied if and only if $x = z = 0$, that is if and only if both variables are *false*, or, in other words, it is satisfied anytime at least one of the variables is *true*. The monotone Disjunctive (mD) kernel [19], $\kappa^{d}_{\text{mD}}(\boldsymbol{x}, \boldsymbol{z})$ counts how many monotone disjunctions, of a fixed arity d, are satisfied in both $\boldsymbol{x}$ and $\boldsymbol{z}$. The embedding of this kernel is defined by Boolean formulas taken from $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = \bigvee_{j \in [\mathcal{U}]^{c}_{i}} x_j\}$, with a feature space of dimension $\binom{n}{d}$. To explain how to count the common positive disjunctions in both $\boldsymbol{x}$ and $\boldsymbol{z}$, we can rely on the analogy between binary vectors and sets. An active disjunction of d literals for $\mathcal{X}$ can be defined as a set of d elements taken from the universe $\mathcal{U}$, let us call it $\mathcal{U}_d$, such that $\exists\ a \in \mathcal{U}_d \mid a \in \mathcal{X}$. Anytime $\exists\ a, b \in \mathcal{U}_d \mid a \in \mathcal{X} \wedge b \in \mathcal{Z}$ (potentially $a = b$), then $\mathcal{U}_d$ is an *active subset* for $\mathcal{X}$ and $\mathcal{Z}$. Using this interpretation, we can say that the value of the kernel is the number of active subsets $\mathcal{U}_d$ in common between $\mathcal{X}$ and $\mathcal{Z}$. We can count the number of these subsets in a negative fashion. Starting from the number of all possible subsets, which is $\binom{|\mathcal{U}|}{d}$, we remove the inactive subsets for $\mathcal{X}$ and for $\mathcal{Z}$. An inactive subset for $\mathcal{X}$ is a set such that it does not contain any element of $\mathcal{X}$, and the number of this kind of sets is $\binom{|\mathcal{U} \setminus \mathcal{X}|}{d}$. Analogously, we can do the same for $\mathcal{Z}$. Now, we have removed twice the subsets formed by elements taken from $\overline{\mathcal{X} \cup \mathcal{Z}} \equiv \mathcal{U} \setminus (\mathcal{X} \cup \mathcal{Z})$ and hence we need to add its contribution once, that is $\binom{|\mathcal{U} \setminus (\mathcal{X} \cup \mathcal{Z})|}{d}$. We can now define κ^{d}_{mD} as

$$\begin{aligned} \kappa^{d}_{\text{mD}}(\boldsymbol{x}, \boldsymbol{z}) &= \binom{|\mathcal{U}|}{d} - \binom{|\mathcal{U} \setminus \mathcal{X}|}{d} - \binom{|\mathcal{U} \setminus \mathcal{Z}|}{d} + \binom{|\mathcal{U} \setminus (\mathcal{X} \cup \mathcal{Z})|}{d} \\ &= \binom{n}{d} - \binom{n - \langle \boldsymbol{x}, \boldsymbol{x} \rangle}{d} - \binom{n - \langle \boldsymbol{z}, \boldsymbol{z} \rangle}{d} + \binom{n - \langle \boldsymbol{x}, \boldsymbol{x} \rangle - \langle \boldsymbol{z}, \boldsymbol{z} \rangle + \langle \boldsymbol{x}, \boldsymbol{z} \rangle}{d}. \end{aligned} \tag{1}$$

3.2. Non-Monotone Boolean Kernels

Converse to the monotone case, *non-monotone* Boolean formulas can contain negated variables, e.g., $\neg x_i$, thus the mL-kernel is not expressive enough to be the simplest non-monotone Boolean kernel because it does not consider negated variables.

3.2.1. Non-Monotone Literal Kernel

To include the contribution of the negated variables, we need to add the number of *false* variables in common between $\boldsymbol{x}$ and $\boldsymbol{z}$ to the mL-kernel. This can be calculated by the *negation* kernel, defined as

$$\begin{aligned} \kappa_{\text{NEG}}(\boldsymbol{x}, \boldsymbol{z}) &= \langle (\mathbf{1}_n - \boldsymbol{x}), (\mathbf{1}_n - \boldsymbol{z}) \rangle \\ &= n - \langle \boldsymbol{x}, \boldsymbol{x} \rangle - \langle \boldsymbol{z}, \boldsymbol{z} \rangle + \langle \boldsymbol{x}, \boldsymbol{z} \rangle, \end{aligned}$$

in which the embedding is defined by Boolean functions taken from $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = \neg x_i\}$. Hence, the non-monotone Literal (L) kernel, $\kappa_L(\boldsymbol{x}, \boldsymbol{z})$, counts how many *true* and *false* variables $\boldsymbol{x}$ and $\boldsymbol{z}$ have in common, and it is defined as a sum of kernels [11] as in the following:

$$\begin{aligned}\kappa_L(\boldsymbol{x}, \boldsymbol{z}) &= \kappa_{\text{mL}}(\boldsymbol{x}, \boldsymbol{z}) + \kappa_{\text{NEG}}(\boldsymbol{x}, \boldsymbol{z}) \\ &= n - \langle \boldsymbol{x}, \boldsymbol{x} \rangle - \langle \boldsymbol{z}, \boldsymbol{z} \rangle + 2\langle \boldsymbol{x}, \boldsymbol{z} \rangle.\end{aligned}$$

3.2.2. Non-Monotone Conjunctive Kernel

The non-monotone Conjunctive (C) kernel counts how many *non-monotone* conjunctions of a certain arity c are satisfied in both $\boldsymbol{x}$ and $\boldsymbol{z}$. The embedding is defined by boolean functions in the set $\mathbb{B}$ of all the non-monotone conjunctions of c literals. Formally, given the set $\mathcal{U}_c^* \equiv \{\mathcal{S} \subseteq \{1, \ldots, 2n\} \mid |\mathcal{S}| = c, \nexists i, j \in \mathcal{S} \text{ s.t. } i = 2j\}$ and the function $g(\boldsymbol{x}, i) = \boldsymbol{x}_i$ if $i \leq n$ or $g(\boldsymbol{x}, i) = \neg \boldsymbol{x}_i$ otherwise, then $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = \bigwedge_{j \in [\mathcal{U}_c^*]_i} g(\boldsymbol{x}, j)\}$. Since we are considering conjunctions of variables, this kernel will count how many combinations of (possibly negated) common variables there are between $\boldsymbol{x}$ and $\boldsymbol{z}$. Thus, relying on these considerations and on the definition of κ_L, we can finally define the $\kappa_{\text{C}}^c(\boldsymbol{x}, \boldsymbol{z})$ as follows:

$$\kappa_{\text{C}}^c(\boldsymbol{x}, \boldsymbol{z}) = \binom{\kappa_L(\boldsymbol{x}, \boldsymbol{z})}{c} = \binom{n - \langle \boldsymbol{x}, \boldsymbol{x} \rangle - \langle \boldsymbol{z}, \boldsymbol{z} \rangle + 2\langle \boldsymbol{x}, \boldsymbol{z} \rangle}{c}.$$

Similar to the monotone case, we can express the Sadohara DNF Kernel [3] as a linear combination of C-kernels of arity $1 \leq d \leq n$ as in the following:

$$\begin{aligned}\kappa_{\text{dnf}}(\boldsymbol{x}, \boldsymbol{z}) &= 2^{\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \overline{\boldsymbol{x}}, \overline{\boldsymbol{z}} \rangle} - 1 = \sum_{i=0}^{n} \binom{\langle \boldsymbol{x}, \boldsymbol{z} \rangle + \langle \overline{\boldsymbol{x}}, \overline{\boldsymbol{z}} \rangle}{i} - 1 = \sum_{i=0}^{n} \binom{\kappa_{\text{mL}}(\boldsymbol{x}, \boldsymbol{z}) + \kappa_{\text{NEG}}(\boldsymbol{x}, \boldsymbol{z})}{i} - 1 \\ &= \sum_{i=0}^{n} \binom{\kappa_L(\boldsymbol{x}, \boldsymbol{z})}{d} - 1 = \sum_{i=1}^{n} \kappa_{\text{C}}^d(\boldsymbol{x}, \boldsymbol{z}).\end{aligned}$$

An analogous construction also holds for the d-DNF kernel.

3.2.3. Non-Monotone Disjunctive Kernel

The non-monotone Disjunctive (D) kernel counts how many *non-monotone* disjunctions, of a certain arity d, are satisfied in both $\boldsymbol{x}$ and $\boldsymbol{z}$. The embedding is defined by Boolean formulas in the set $\mathbb{B}$ of all the non-monotone disjunctions of d literals. Formally, $\mathbb{B} \equiv \{f_i \mid f_i(\boldsymbol{x}) = \bigvee_{j \in [\mathcal{U}_d^*]_i} g(\boldsymbol{x}, j)\}$, with $\mathcal{U}_d^*$ and g defined as in the previous section. As for the *monotone* case, we derive the kernel function in a negative fashion. The number of every possible combination of arity d of variables that can be also in their negated form is $2^d\binom{n}{d}$. For both $\boldsymbol{x}$ and $\boldsymbol{z}$, we have to discard the combinations that are *false*, which are exactly $\binom{n}{d}$ because for each set of d different variables, there exists only one assignment of the negations such that the disjunction is *false*. For example, given the variables $x_1 = 1, x_2 = 0$ and $x_3 = 1$, only the disjunction $\neg x_1 \vee x_2 \vee \neg x_3$ is *false* and all the others $2^3 - 1$ negation assignments are *true*. Then, we have to re-add the *false* combinations that have been discarded twice, that is the combinations made with variables that are *false* in both the vectors. This can be seen as the opposite of what C-kernel computes, but, since we generate all the possible combinations with all the possible negation assignments, the counting is actually the same as the C-kernel. Finally, we can define the $\kappa_{\text{D}}^d(\boldsymbol{x}, \boldsymbol{z})$ as

$$\kappa_{\text{D}}^d(\boldsymbol{x}, \boldsymbol{z}) = (2^d - 2)\binom{n}{d} + \kappa_{\text{C}}^d(\boldsymbol{x}, \boldsymbol{z})$$

3.3. Boolean Kernels Combination

Given the Boolean kernels defined in the previous sections, we have now all the basic elements to build new Boolean kernels that represent a specific Boolean concept. Table 1 shows a summary of

all the presented Boolean kernels. It is easy to see that all the kernels are function of dot products of the input vectors, and this allows us to create new kernels by replacing those dot products with other Boolean kernels. The logical interpretation of the new kernel depends on how the base kernels are combined. In the following, we present some new Boolean kernels generated by using the above mentioned method.

Table 1. Summary of the just presented Boolean kernels: $\boldsymbol{x}, \boldsymbol{z} \in \{0,1\}^n$ and $|\phi|$ stands for the dimension of the feature space.

κ	$\kappa(\boldsymbol{x}, \boldsymbol{z})$	$\kappa(\boldsymbol{x}, \boldsymbol{x})$	$\lvert\phi\rvert$
κ_{mL}	$\langle \boldsymbol{x}, \boldsymbol{z} \rangle$	$\lvert\boldsymbol{x}\rvert$	n
κ^c_{mC}	$\binom{\kappa_{\text{mL}}(\boldsymbol{x},\boldsymbol{z})}{c}$	$\binom{\lvert\boldsymbol{x}\rvert}{c}$	$\binom{n}{c}$
κ^d_{mD}	$\binom{n}{d} - \binom{n-\lvert\boldsymbol{x}\rvert}{d} - \binom{n-\lvert\boldsymbol{z}\rvert}{d} + \binom{n-\lvert\boldsymbol{x}\rvert-\lvert\boldsymbol{z}\rvert+\langle\boldsymbol{x},\boldsymbol{z}\rangle}{d}$	$\binom{n}{d} - \binom{n-\lvert\boldsymbol{x}\rvert}{d}$	$\binom{n}{d}$
κ_{NEG}	$n - \lvert\boldsymbol{x}\rvert - \lvert\boldsymbol{z}\rvert + \langle \boldsymbol{x}, \boldsymbol{z} \rangle$	$n - \lvert\boldsymbol{x}\rvert$	n
κ_L	$\kappa_{\text{mL}} + \kappa_{\text{NEG}}$	n	$2n$
κ^c_{C}	$\binom{\kappa_L(\boldsymbol{x},\boldsymbol{z})}{c}$	$\binom{n}{c}$	$2^c\binom{n}{c}$
κ^d_{D}	$(2^d-2)\binom{n}{d} + \kappa^d_{\text{C}}(\boldsymbol{x}, \boldsymbol{z})$	$(2^d-1)\binom{n}{d}$	$2^d\binom{n}{d}$

3.3.1. DNF Kernels

A disjunctive normal form (DNF) is a normalization of a logical formula which is a disjunction of conjunctive clauses. In the monotone case, both conjunctive and disjunctive clauses are monotone, i.e., the literals are only in their positive form. Since DNFs are disjunctions of conjunctive clauses, we can combine the embedding maps of the mC-kernel and the mD-kernel in this way $\phi^{d,c}_{\text{mDNF}} : \boldsymbol{x} \mapsto \phi^d_{\text{mD}}(\phi^c_{\text{mC}}(\boldsymbol{x}))$, obtaining the desired feature space for the monotone DNF, which leads to the definition of the mDNF-kernel as in the following:

$$\kappa^{d,c}_{\text{mDNF}}(\boldsymbol{x}, \boldsymbol{z}) = \binom{\binom{n}{c}}{d} - \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\boldsymbol{x}, \boldsymbol{x})}{d} - \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\boldsymbol{z}, \boldsymbol{z})}{d} + \binom{\binom{n}{c} - \kappa^c_{\text{mC}}(\boldsymbol{x}, \boldsymbol{x}) - \kappa^c_{\text{mC}}(\boldsymbol{z}, \boldsymbol{z}) + \kappa^c_{\text{mC}}(\boldsymbol{x}, \boldsymbol{z})}{d}.$$

Note that we need to know the dimension of the feature space of the mC-kernel. The features of this kernel are actually monotone DNF formulas with exactly d disjunctions of conjunctive clauses of arity c. In the non-monotone case, we proceed in a similar way but, since the conjunctive clauses are non-monotone, we have to use the C-kernel instead of the mC-kernel. So, the feature space can be created by composing the embedding map of the mD-kernel with the one of the C-kernel, that is, $\phi^{d,c}_{\text{DNF}} : \boldsymbol{x} \mapsto \phi^d_{\text{mD}}(\phi^c_{\text{C}}(\boldsymbol{x}))$. Consequently, the DNF-kernel is defined as follows:

$$\kappa^{d,c}_{\text{DNF}}(\boldsymbol{x}, \boldsymbol{z}) = \binom{2^c\binom{n}{c}}{d} - \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\boldsymbol{x}, \boldsymbol{x})}{d} - \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\boldsymbol{z}, \boldsymbol{z})}{d} + \binom{2^c\binom{n}{c} - \kappa^c_{\text{C}}(\boldsymbol{x}, \boldsymbol{x}) - \kappa^c_{\text{C}}(\boldsymbol{z}, \boldsymbol{z}) + \kappa^c_{\text{C}}(\boldsymbol{x}, \boldsymbol{z})}{d}.$$

3.3.2. CNF Kernels

A logic formula is in conjunctive normal form (CNF) if it is composed of conjunctions of disjunctive clauses. Clearly, in its monotone form it does not contain any negated literal. By using a similar approach as for the mDNF-kernel, the feature space is defined by the function $\phi^{d,c}_{\text{mCNF}} : \boldsymbol{x} \mapsto \phi^d_{\text{mC}}(\phi^c_{\text{mD}}(\boldsymbol{x}))$, and hence in the kernel function we replace the dot products inside the mC-kernel with the mD-kernel as in the following:

$$\kappa^{d,c}_{\text{mCNF}}(\boldsymbol{x}, \boldsymbol{z}) = \binom{\kappa^d_{\text{mD}}(\boldsymbol{x}, \boldsymbol{z})}{c}.$$

The resulting feature space is composed of monotone CNF formulas with exactly c conjunctions of disjunctive clauses of arity d. By swapping the mD-kernel with the D-kernel, we can easily obtain the non-monotone CNF kernel:

$$\kappa_{\text{CNF}}(\boldsymbol{x}, \boldsymbol{z}) = \binom{\kappa_{\text{D}}^{d}(\boldsymbol{x}, \boldsymbol{z})}{c},$$

which has associated the embedding function $\phi_{\text{CNF}}^{d,c} : \boldsymbol{x} \mapsto \phi_{\text{mC}}^{d}(\phi_{\text{D}}^{c}(\boldsymbol{x}))$,

For the sake of brevity, in the rest of the paper, we indicate the mDNF-kernel having d disjunctions and c conjunctions with either the notation mDNF(d,c)-kernel or simply mDNF(d,c).

4. Computational Complexity

The computational complexity of the Boolean kernels described in the previous sections is bounded by the complexity of the calculation of the binomial coefficient, that is $\mathcal{O}(k)$ with k the arity of the combinations. Hence, computing an entire kernel matrix over n-dimensional examples taken from a dataset with l examples would lead to a complexity of $\mathcal{O}((k+n) \cdot l^2)$. Even though it is not possible to reduce such complexity, we can take advantage of the recursive nature of the binomial coefficient to compute the kernels in a recursive fashion. By doing so, it is possible to compute higher degree kernel matrices by leveraging on kernel matrices of lower degrees.

Let the matrix $\boldsymbol{K}^0$ be the base (Boolean) kernel matrix over the dataset $\mathcal{D}$, such that $\boldsymbol{K}^0_{i,j} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$, for $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{D}$ and some ϕ with codomain $\{0,1\}^n$. Then, we can recursively define the Boolean kernels for both monotone and the non-monotone case as described in the following.

4.1. mC-Kernel

By definition, the mC-kernel matrix of arity 1, that is $\boldsymbol{K}^1_{\text{mC}}$, is equivalent to the base kernel matrix $\boldsymbol{K}^0$. By using $\boldsymbol{K}^0$ as base case, we can recursively define the mC-kernel matrix as

$$\boldsymbol{K}^{c+1}_{\text{mC}} = \boldsymbol{K}^{c}_{\text{mC}} \odot \left(\frac{1}{c+1} \left(\boldsymbol{K}^{1}_{\text{mC}} - c \mathbf{1}_l \mathbf{1}_l^\intercal \right) \right).$$

4.2. mD-Kernel

Let us define the matrices

$$\mathbf{S} = \text{diag}(\boldsymbol{K}^0) \cdot \mathbf{1}_l^\intercal, \quad \mathbf{N}_x = n \mathbf{1}_l \mathbf{1}_l^\intercal - \mathbf{S}, \text{ and } \mathbf{N}_{xz} = \mathbf{N}_x - \mathbf{S}^\intercal + \boldsymbol{K}^0.$$

Then, we define, recursively in its parts, the mD-kernel matrix $\boldsymbol{K}^{d}_{\text{mD}}$ as

$$\boldsymbol{K}^{d}_{\text{mD}} = \mathbf{N}^{(d)} - \mathbf{N}_x^{(d)} - \mathbf{N}_x^{(d)\intercal} + \mathbf{N}_{xz}^{(d)},$$

where

$$\begin{aligned}
\mathbf{N}^{(d+1)} &= \mathbf{N}^{(d)} \odot \left(\frac{n-d}{d+1} \mathbf{1}_l \mathbf{1}_l^\intercal \right), \\
\mathbf{N}_x^{(d+1)} &= \mathbf{N}_x^{(d)} \odot \left(\frac{1}{d+1} \left(\mathbf{N}_x - d \mathbf{1}_l \mathbf{1}_l^\intercal \right) \right), \text{ and} \\
\mathbf{N}_{xz}^{(d+1)} &= \mathbf{N}_{xz}^{(d)} \odot \left(\frac{1}{d+1} \left(\mathbf{N}_{xz} - d \mathbf{1}_l \mathbf{1}_l^\intercal \right) \right),
\end{aligned}$$

with the corresponding base cases $\mathbf{N}^{(1)} = n \mathbf{1}_l \mathbf{1}_l^\intercal$, $\mathbf{N}_x^{(1)} = \mathbf{N}_x$ and $\mathbf{N}_{xz}^{(1)} = \mathbf{N}_{xz}$.

4.3. C-Kernel

By relying on the previous definition of $\mathbf{S}$ and the base case kernel matrix

$$\boldsymbol{K}_{\mathrm{C}}^{1} = n\mathbf{1}_l\mathbf{1}_l^{\intercal} - \mathbf{S} - \mathbf{S}^{\intercal} + 2\boldsymbol{K}^{0},$$

the C-kernel matrix can be recursively defined by

$$\boldsymbol{K}_{\mathrm{C}}^{c+1} = \boldsymbol{K}_{\mathrm{C}}^{c} \odot \left(\frac{1}{c+1} \left(\boldsymbol{K}_{\mathrm{C}}^{1} - c\mathbf{1}_l\mathbf{1}_l^{\intercal} \right) \right).$$

4.4. D-Kernel

Using the previous definitions of the matrices $\mathbf{N}^{(d)}$ and $\boldsymbol{K}_{\mathrm{C}}^{c}$, we can define the D-kernel matrix, recursively in its parts, as

$$\boldsymbol{K}_{\mathrm{D}}^{d+1} = \left((2^d - 2)\mathbf{1}_l\mathbf{1}_l^{\intercal} \right) \odot \mathbf{N}^{(d)} - \boldsymbol{K}_{\mathrm{C}}^{d}.$$

Both the standard and the recursive definition have been implemented in a freely available Python module *pyros* available at the following URL: https://github.com/makgyver/pyros.

5. Evaluation

5.1. Evaluation Protocol

In all experiments, the kernels have been normalized using the well-known formula

$$\tilde{\kappa}(\boldsymbol{x}, \boldsymbol{z}) = \frac{\kappa(\boldsymbol{x}, \boldsymbol{z})}{\sqrt{\kappa(\boldsymbol{x}, \boldsymbol{x})\kappa(\boldsymbol{z}, \boldsymbol{z})}}.$$

We assessed the proposed Boolean kernels by using SVM as kernel machine and we compared them with the linear kernel, the RBF kernel, the (monotone) DNF kernel proposed by Sadohara et al. [3], the d-DNF kernel [9], the σ-mDNF kernel [10] and the Tanimoto kernel.

Both validation and test were evaluated in terms of the Area Under the ROC Curve (AUC). The used validation method is a five-fold nested cross validation. For each dataset, the test was repeated 10 times and the average performances were recorded. Specifically, we validated the misclassification cost parameter C for the SVM in the set $\{2^{-4}, 2^{-3}, \ldots, 2^{4}\}$; for each of the proposed kernels we validated both the conjunctive arity c (when applicable) and the disjunctive arity d (when applicable) in the set $\{1, \ldots, 5\}$, for the RBF kernel we validated the hyper-paramater $\gamma \in \{10^{-4}, \ldots, 10^{4}\}$, while for the d-mDNF we fixed $d = 5$. Finally, for the σ-mDNF kernel we validated σ in the set $\{0.2, 0.5, 1, 2\}$. All the experiments were implemented in Python using the machine learning module Scikit-learn [20]. The source code is freely available at https://github.com/makgyver/pyros.

The benchmark datasets used for the experiments are reported in Table 2. These datasets are freely available from the UCI repository [21] and the KEEL repository [22]. We selected datasets with binary or categorical features and for each of them the following preprocessing steps were performed:

- Instances with missing attributes were removed.
- categorical features, including the binary ones, were mapped into binary features by means of the *one-hot* encoding [23]. This preprocessing keeps for every example in the dataset the same number of ones , or in other words every input vector has the same euclidean norm.
- Non-binary tasks were artificially transformed into binary ones, by arranging the classes into two groups while trying to keep the number of instances balanced.

Table 2. Datasets information: name, number of instances (# Examples), number of features (# Features), class distribution and number of active variables for example.

Dataset Name	# Examples	pos/neg (%)	# Features	$m = \|x\|_1$
zoo	101	40/60	36	16
promoters	106	50/50	228	57
lymphography	148	45/55	44	15
house-votes	232	46/54	32	16
soybean	266	54/46	97	35
spect	267	79/21	44	22
breast	277	71/29	41	9
primary-tumor	339	41/59	34	15
monks-1	432	50/50	17	6
crx	653	55/45	40	9
tic-tac-toe	958	65/35	27	9
flare	1066	49/51	41	11
car	1728	30/70	21	6
dna_bin	2000	47/53	180	47
splice	3175	48/52	240	60
kr-vs-kp	3196	52/48	73	36

5.2. Experimental Results

The first set of experiments assessed the quality of the proposed kernels. The average AUCs over 10 runs as well as the standard deviations are reported in Table 3. The last row of each table summarizes the average rank achieved by all kernels over the benchmark datasets.

It is evident from the tables that normal form (NF) kernels perform on average better than the other Boolean kernels, with the only exception of the C-kernel on the AUC metric. This is reasonable since normal form kernels are a generalization of the (m)C-kernel and the (m)D-kernel. However, after the validation procedure, there is no guarantee that a NF kernel is always better than their correspondent base kernels, as underlined by the very good performance of the C-kernel. Another interesting observation is that both the D-kernel and the mD-kernel are almost always the worst performing ones, and this can be explained by the fact that they are less expressive than the competing kernels.

Since NF kernels have shown very good performances, we built, for each normal form, the average kernel over all the degrees, that is:

$$\kappa_{\text{avg.NF}} = \frac{1}{C \times D} \sum_{d=1}^{D} \sum_{c=1}^{C} \kappa_{NF}^{d,c}(\boldsymbol{x}, \boldsymbol{z}).$$

In this way, given a normal form, the feature space of the resulting kernel contains all the normal form formulas for each of the degrees (c, d) with $1 \leq c \leq C$ and $1 \leq d \leq D$. Since we have fixed the maximum degrees C and D or all the kernels, and the $\kappa_{\text{avg.NF}}$ has no other hyper-parameters, it did not require any further validation.

Table 3. AUC performances on benchmark datasets. For each dataset the best performing kernel is highlighted with both the **boldface** font and with a dot (·).

Dataset	mC	mD	C	D	mDNF	mCNF	DNF	CNF
breast	71.19 ±1.27	**71.68**· ±1.51	71.52 ±1.87	71.70 ±1.80	71.21 ±1.73	71.31 ±2.07	71.54 ±1.82	71.50 ±1.90
car_bin	**100.00**· ±0.00	99.97 ±0.08	**100.00**· ±0.00	99.62 ±0.12	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
crx	91.77 ±0.39	92.04 ±0.37	**92.11**· ±0.22	91.99 ±0.28	91.81 ±0.34	91.86 ±0.38	92.08 ±0.29	92.03 ±0.30
dna_bin	99.03 ±0.05	98.98 ±0.06	**99.07**· ±0.05	98.96 ±0.05	**99.07**· ±0.05	99.01 ±0.04	**99.07**· ±0.05	99.06 ±0.05
flare_bin	94.80 ±0.13	94.91 ±0.13	94.91 ±0.13	94.91 ±0.10	94.88 ±0.11	94.89 ±0.11	**94.92**· ±0.10	94.90 ±0.13
house-votes-84	99.23 ±0.26	99.19 ±0.28	**99.27**· ±0.25	99.18 ±0.31	99.19 ±0.28	99.18 ±0.29	99.18 ±0.30	99.16 ±0.32
kr-vs-kp	99.95 ±0.02	99.74 ±0.01	99.94 ±0.02	99.74 ±0.01	**99.96**· ±0.01	**99.96**· ±0.01	**99.96**· ±0.01	**99.96**· ±0.01
lymphography_bin	92.45 ±1.45	92.81 ±1.09	92.77 ±1.10	92.73 ±1.23	92.40 ±1.33	92.67 ±1.25	92.86 ±1.13	**93.03**· ±1.16
monks-1	**100.00**· ±0.00	91.58 ±1.40	**100.00**· ±0.00	87.75 ±1.86	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
primary-tumor	72.88 ±0.61	72.92 ±0.56	72.80 ±0.79	72.72 ±0.58	**72.97**· ±0.54	72.95 ±0.51	72.70 ±0.82	72.86 ±0.61
promoters	97.51 ±1.06	97.39 ±0.92	97.39 ±0.87	97.28 ±0.87	**97.57**· ±1.09	97.49 ±1.13	97.43 ±0.89	97.45 ±0.88
soybean	99.66 ±0.08	99.64 ±0.09	**99.69**· ±0.09	99.66 ±0.07	99.66 ±0.10	99.65 ±0.09	99.68 ±0.07	**99.69**· ±0.06
spect	83.60 ±2.07	83.58 ±2.08	83.79 ±1.98	83.75 ±2.06	83.61 ±2.04	83.63 ±2.04	**83.93**· ±2.00	83.81 ±2.08
splice	**99.35**· ±0.04	99.20 ±0.04	99.28 ±0.02	99.19 ±0.04	**99.35**· ±0.03	**99.35**· ±0.04	99.29 ±0.02	99.29 ±0.02
tic-tac-toe	98.03 ±0.61	97.86 ±0.43	**98.25**· ±0.37	**98.25**· ±0.37	98.03 ±0.60	98.03 ±0.61	**98.25**· ±0.37	**98.25**· ±0.37
zoo	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
Rank	4.38	5.13	3.06	5.44	4.00	3.88	**2.88**	3.13

The average AUCs over the 10 runs as well as the standard deviations are reported in Table 4.

In general, we can see that the normal form kernels seem to achieve performances almost always comparable or higher than the competing kernels. Good performance is also achieved by the d-mDNF kernel which we have shown is the summation of d mC-kernels.

An interesting observation can be done regarding the monks-1 dataset which can be explained by a mDNF rule. The obtaining results on monks-1 show that most of the proposed Boolean kernels achieve an AUC of 100% while all other kernels are not able to achieve this perfect score. This is because these Boolean kernels contain the target formula, or a set of related formulas, in the feature space.

All the reported results are further confirmed by other experiments with other metrics, such as precision, recall and F1, however they are not reported here for space reasons.

Table 4. AUC performances on benchmark datasets. For each dataset the best performing kernel is highlighted with both the **boldface** font and with a dot (·).

Dataset	Linear	RBF	DNF [3]	d-mDNF	Tanimoto	σ-mDNF	avg.mDNF	avg.mCNF	avg.DNF	avg.CNF
breast	70.69 ±0.91	71.21 ±1.63	71.56 ±1.55	71.82 ±1.50	**72.32**· ±1.99	71.24 ±1.77	70.09 ±1.69	72.16 ±1.85	72.18 ±1.97	72.03 ±1.73
car_bin	98.98 ±0.13	99.96 ±0.07	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	99.82 ±0.04	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
crx	91.08 ±0.41	92.03 ±0.36	91.96 ±0.21	91.82 ±0.23	**92.14**· ±0.26	91.99 ±0.36	91.78 ±0.21	91.96 ±0.30	92.02 ±0.27	92.09 ±0.25
dna_bin	98.21 ±0.10	98.84 ±0.06	97.25 ±0.19	98.50 ±0.05	98.92 ±0.05	98.52 ±0.06	98.51 ±0.05	99.04 ±0.05	99.07 ±0.05	**99.07**· ±0.05
flare	94.85 ±0.14	**94.87**· ±0.14	93.72 ±0.18	94.53 ±0.12	94.84 ±0.17	94.82 ±0.10	93.89 ±0.17	94.76 ±0.12	94.76 ±0.18	94.86 ±0.16
house-v.	**99.38**· ±0.18	99.36 ±0.17	98.67 ±0.24	98.96 ±0.26	99.26 ±0.26	99.16 ±0.21	98.85 ±0.23	98.87 ±0.20	98.94 ±0.25	98.94 ±0.25
kr-vs-kp	99.14 ±0.01	**99.98**· ±0.01	99.94 ±0.01	99.97 ±0.01	99.94 ±0.01	**99.98**· ±0.01	99.97 ±0.01	99.94 ±0.01	99.97 ±0.01	99.94 ±0.01
lympho.	92.37 ±1.20	92.62 ±1.28	92.20 ±0.82	93.08 ±1.01	93.03 ±1.10	93.10 ±1.17	92.86 ±0.89	93.07 ±1.02	93.13 ±1.01	**93.34**· ±1.08
monks-1	46.32 ±3.12	99.10 ±0.47	89.87 ±1.52	91.98 ±1.35	99.70 ±0.23	99.71 ±0.32	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
primary-t.	73.31 ±0.81	72.75 ±0.90	73.01 ±0.89	**73.41**· ±0.39	73.00 ±0.68	72.99 ±0.54	73.37 ±0.55	73.06 ±0.54	73.12 ±0.45	73.34 ±0.45
promoters	97.23 ±0.98	97.37 ±0.98	95.07 ±0.88	**97.65**· ±0.82	97.31 ±0.94	97.29 ±0.92	97.61 ±0.75	97.46 ±0.91	97.37 ±0.92	97.41 ±0.83
soybean	99.47 ±0.12	99.68 ±0.10	99.56 ±0.08	99.72 ±0.07	99.65 ±0.06	99.74 ±0.10	**99.73**· ±0.09	99.72 ±0.07	99.71 ±0.09	99.71 ±0.09
spect	**83.81**· ±1.82	83.70 ±1.82	78.57 ±1.90	82.34 ±1.94	84.14 ±1.92	83.43 ±1.77	82.27 ±1.97	82.25 ±2.04	82.53 ±1.99	82.42 ±1.95
splice	98.48 ±0.03	99.12 ±0.04	98.27 ±0.13	**99.31**· ±0.04	99.11 ±0.04	99.01 ±0.05	99.30 ±0.04	99.25 ±0.03	99.29 ±0.03	99.28 ±0.03
t-t-t	97.86 ±0.43	98.53 ±0.18	99.94 ±0.03	**99.97**· ±0.02	99.89 ±0.05	99.88 ±0.05	99.96 ±0.03	99.95 ±0.03	99.96 ±0.03	99.95 ±0.02
zoo	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00	**100.00**· ±0.00
Rank	7.25	5.25	7.81	4.13	5.00	5.13	5.31	4.56	3.50	**3.44**

6. Conclusions

We present a family of Boolean kernels designed to have feature spaces composed by logical formulas that can be exploited to interpret the solution of a large-margin kernel machine, such as an SVM. For all kernels, we provide the logic interpretation and an efficient way to compute them. Experimental results on many categorical datasets show that the presented kernels achieve a performance comparable to the performance of state-of-the-art kernels (such as the RBF) and other Boolean kernels. In particular, we observed that, in general, those kernels corresponding to normal form achieve very good performances. In the future, we aim to develop methods able to learn how to combine different Boolean kernels, for example by means of Multiple Kernel Learning algorithms. Moreover, we also aim to find efficient and effective ways to interpret the solution of SVMs based on Boolean kernels.

Author Contributions: M.P. and F.A. contributed to the development of the Boolean kernel framework. M.P. conceived, designed and performed the experiments; I.L. collected and prepared the datasets. M.P. wrote the paper and F.A. revised the draft.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barakat, N.; Bradley, A.P. Rule extraction from support vector machines: A review. *Neurocomputing* **2010**, *74*, 178–190.
2. Polato, M.; Lauriola, I.; Aiolli, F. Classification of Categorical Data in the Feature Space of Monotone DNFs. In Proceedings of the 2017 International Conference on Artificial Neural Networks and Machine Learning, Alghero (Sardinia), Italy, 11–14 September 2017.
3. Sadohara, K. Learning of Boolean Functions Using Support Vector Machines. In Proceedings of the 12th International Conference on Algorithmic Learning Theory, Washington, DC, USA, 25–28 November 2001; Abe, N., Khardon, R., Zeugmann, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 106–118.
4. Kimeldorf, G.S.; Wahba, G. Some Results on Tchebycheffian Spline Functions. *J. Math. Anal. Appl.* **1971**, *33*, 82–95.
5. Hofmann, T.; Schölkopf, B.; Smola, A.J. Kernel methods in machine learning. *Ann. Stat.* **2008**, *36*, 1171–1220.
6. Watkins, C. *Kernels from Matching Operations*; Technical Report; Department of Computer Science, Royal Holloway, University of London: London, UK, 1999.
7. Khardon, R.; Roth, D.; Servedio, R. Efficiency Versus Convergence of Boolean Kernels for On-line Learning Algorithms. In Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, Vancouver, BC, Canada, 3–8 December 2001; MIT Press: Cambridge, MA, USA, 2001; pp. 423–430.
8. Sadohara, K. On a Capacity Control Using Boolean Kernels for the Learning of Boolean Functions. In Proceedings of the 2002 IEEE International Conference on Data Mining, Maebashi City, Japan, 9–12 December 2002; IEEE Computer Society: Washington, DC, USA, 2002; pp. 410–417.
9. Nguyen, S.H.; Nguyen, H.S. Applications of Boolean Kernels in Rough Sets. In Proceedings of the Second International Conference on Rough Sets and Intelligent Systems Paradigms, Granada and Madrid, Spain, 9–13 July 2014; pp. 65–76.
10. Zhang, Y.; Li, Z.; Kang, M.; Yan, J. Improving the classification performance of boolean kernels by applying Occam's razor. In Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS '03), Singapore, 15–18 December 2003.
11. Shawe-Taylor, J.; Cristianini, N. *Kernel Methods for Pattern Analysis*; Cambridge University Press: New York, NY, USA, 2004.
12. Kusunoki, Y.; Tanino, T. Boolean kernels and clustering with pairwise constraints. In Proceedings of the 2014 IEEE International Conference on Granular Computing (GrC), Noboribetsu, Japan, 22–24 October 2014; pp. 141–146.
13. Kowalczyk, A.; Smola, A.J.; Williamson, R.C. Kernel Machines and Boolean Functions. In *Advances in Neural Information Processing Systems 14, Proceedings of the Neural Information Processing Systems, Natural and Synthetic, Vancouver, BC, Canada, 3–8 December 2001*; MIT Press: Cambridge, MA, USA, 2001; pp. 439–446.

14. Khardon, R.; Servedio, R.A. Maximum Margin Algorithms with Boolean Kernels. In *Learning Theory and Kernel Machines, Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, 24–27 August 2003*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 87–101.
15. Cui, K.; Han, F.; Wang, P. Research on Face Recognition Based on Boolean Kernel SVM. In Proceedings of the 2008 Fourth International Conference on Natural Computation, Jinan, China, 18–20 October 2008; Volume 2, pp. 148–152.
16. Cui, K.; Du, Y. Application of Boolean Kernel Function SVM in Face Recognition. In Proceedings of the 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, China, 25–26 April 2009; Volume 1, pp. 619–622.
17. Liu, S.; Cui, K. Applications of Support Vector Machine Based on Boolean Kernel to Spam Filtering. *Modern Appl. Sci.* **2009**, *3*, 27.
18. Cui, K.; Du, Y. Short-Term Load Forecasting Based on the BKF-SVM. In Proceedings of the 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, China, 25–26 April 2009; Volume 2, pp. 528–531.
19. Polato, M.; Aiolli, F. Boolean kernels for collaborative filtering in top-N item recommendation. *Neurocomputing* **2018**, *286*, 214–225.
20. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
21. Lichman, M. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/ (accessed on 28 February 2018).
22. Alcalá-Fdez, J.; Fernández, A.; Luengo, J.; Derrac, J.; García, S.; Sánchez, L.; Herrera, F. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Logic Soft Comput.* **2011**, *17*, 255–287.
23. Harris, D.M.; Harris, S.L. *Digital Design and Computer Architecture*, 2nd ed.; Morgan Kaufmann: Boston, MA, USA, 2013.