PRL: A game theoretic large margin method for interpretable feature learning

PRL: A game theoretic large margin method for interpretable feature learning

Mirko Polato[a,*], Guglielmo Faggioli[b], Fabio Aiolli[a]

[a]*Department of Mathematics, University of Padova, Via Trieste, 63, 35121 Padova, Italy.*
[b]*Department of Engineering, University of Padova, Via Gradenigo, 6/b, 35131 Padova, Italy.*

Abstract

The crucial role played by interpretability in many practical scenarios has lead a large part of the research on machine learning towards the development of interpretable approaches. In this work, we present a game-theory based method capable of achieving state-of-the-art accuracy, yet keeping the focus on the interpretability of the predictions. The proposed approach is an instance of the more general preference learning framework. By design, the method identifies the most relevant features even when dealing with high-dimensional problems. This is possible thanks to an online features generation mechanism. Moreover, the algorithm is proven to be theoretically well-founded, thanks to a game theoretical analysis on its convergence. To assess the quality of the proposed approach, it has been compared against state-of-the-art methods in a plethora of different classification settings. The experimental evaluation focuses on interpretability, with an in-depth analysis on visualization, feature selection and explainability.

Keywords: game theory, preference learning, large margin, feature selection, interpretability

*Corresponding author
Email addresses: mpolato@math.unipd.it (Mirko Polato), guglielmo.faggioli@phd.unipd.it (Guglielmo Faggioli), aiolli@math.unipd.it (Fabio Aiolli)

1. Introduction

Game theory (GT) and computer science have always had a strong bond. Much of the current research in GT dates back to the work of computer science pioneers like, for example, John Von Neumann and Alan Turing. In its early days, artificial intelligence research made an extensive use of games as training environments for the development of novel algorithms. Game theoretical concepts are at the core of many machine learning (ML) approaches: reinforcement learning and imitation learning are two of many possible examples. However, Adversarial learning [1] is by far the hottest ML topic related to GT, since its competitive nature made it the perfect learning framework for applicative areas such as cyber security [2].

Historically, adversarial concepts are also at the basis of the seminal work that introduced Adaboost [3], in which GT is applied to on-line learning. The GT-ML connection has been also extensively studied in the context of Support Vector Machine (SVM). For instance, it can be shown that the hard margin SVM can be cast into a two-players zero-sum game [4, 5].

In this paper, we present a principled algorithm, dubbed PRL (Preference and Rule Learning), inspired by preference learning [6] and game theory. PRL aims at maximizing the minimum margin in the space of preferences represented using the Kessler's construction [7]. In PRL, the learning problem is cast into a two-players zero-sum game, where a player tries to select hypotheses for maximizing the margin, and the opponent chooses adversarial preferences in order to minimize it. The considered hypotheses spaces consist in a set of preference prototypes along with (possibly non-linear) features. One important characteristic of PRL is that, by design, feature selection represents an integral part of the learning process.

To deal with high dimensional data and high dimensional feature spaces, PRL generates features in an on-line fashion. As we will show later, the on-line feature generation plays a very important role in PRL, especially when it comes to interpret the solution of the model, which is very useful for producing explanations.

Nowadays, machine learning methods are widely used by non-practitioners and having the ability of interpreting their model is often desirable. There are plenty of

applications in which explanation plays a key role, such as bioinformatic applications, recommender systems, and support systems for physicians, just to mention a few. Moreover, the notion of explainability of automatic systems is also one of the most controversial subject contained in the recently introduced European regulation GDPR (General Data Protection Regulation). PRL offers the ability to design feature spaces composed by logical rules and thanks to its feature selection capabilities gives the opportunity to interpret the provided prediction.

To summarize, the main contributions of this work are listed in the following:

- a new large margin method based on preference learning and game theoretical concepts for label ranking/multi-class classification. The method naturally comes with feature selection capabilities. PRL is also able to deal with (non linear) feature spaces of infinitely many dimensions, thanks to the online feature generation;
- the framework is general enough to deal with different kinds of features and rules that are very useful when interpretability is desired;
- a theoretical study on the convergence to the optimal solution;
- a parallelized version of *Fictitious Play* [8] for solving game matrices;
- an extensive set of experiments is reported. Experiments have been performed with the aim of assessing different aspects of PRL: (i) effectiveness, (ii) feature selection capability, and (iii) interpretability. Results show that PRL is able to provide sparse solutions that are suitable when the explanation of the decision is desirable.

The work presented here extends the paper [9], in particular:

- we propose a parallelized version of the algorithm *Fictitious Play* for solving game matrices and demonstrate its convergence (Section 3);
- we introduce the decision path rules generation scheme (Section 6.3);
- we also present a dynamic budget version of PRL which automatically changes the columns' budget when needed;

- we add more details and experiments regarding the *bias* feature, showing its usefulness in producing sparser solutions;

- we integrate the evaluation with additional experiments, e.g., using the decision path rules generation scheme (Section 7).

The remainder of this paper is structured as follows: Section 2 will introduce the background knowledge useful to understand the theoretical concepts of the paper. Section 3 presents Parallel Fictitious Play. Section 4 and 5 will present the main contribution of the paper, that is the PRL method. In Section 6 the on-line feature generation used in PRL is presented, providing different feature generation schemes. Section 7 is dedicated to the evaluation of the proposed approach. Section 8 discusses works related to PRL, especially connections between machine learning and game theory, as well as other on-line non-linear feature selection approaches. Finally, Section 9 wraps up the contribution of the paper and discusses possible future works.

2. Background

In this section we present all the necessary notions to understand the rest of the paper. In particular, we introduce both Preference Learning (Section 2.1) and Game Theory (Section 2.2) focussing on the key elements used throughout the paper. Finally, in Section 3 we present a parallel version of the classical Fictitious Play algorithm which is described in sections 2.3 and 3.

2.1. Preference Learning

Preference learning (PL) is a sub-task in machine learning in which the input data consists of preference relations. Such preferences are assumed to be in agreement with some utility function g_θ. In PL problems, the goal is to build a preference model, i.e., find the parameters θ of the utility function g able to predict preferences for previously unseen items. In the context of label ranking, for instance, the training set consists of a set of pairwise preferences $y_i \succ_{\mathbf{x}} y_j$, $i \neq j$, that is, for the pattern $\mathbf{x}$, label y_i is preferred to label y_j.

In this work we consider one of the main PL tasks, that is label ranking [6]: given a set of input patterns $\mathbf{x}_i \in \mathcal{X}$, $i \in [1, \ldots, n]$, and a finite set of labels $\mathcal{Y} \equiv \{y_1, y_2, \ldots, y_m\}$ the goal is to learn a scoring function $g_\theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ which assigns a score for each instance-label pair $(\mathbf{x}, y)$. It is worth to notice that label ranking represents a generalization of a classification task since g_θ implicitly defines, for an instance $\mathbf{x}$, a total order over $\mathcal{Y}$. We focus on linear preference models [10, 11] of the form $g_\theta(\mathbf{x}, y) = \mathbf{w}^\intercal \psi(\mathbf{x}, y)$, where $\theta \equiv \mathbf{w} \in \mathbb{R}^{d \cdot m}$ is the vector of model parameters, and $\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{d \cdot m}$, $\mathcal{X} \equiv \mathbb{R}^d$, $\mathcal{Y} \equiv \{1, \ldots, m\}$ is a joint representation of instance and label pairs.

Given an item, the goal of the model g_θ is to correctly rank the labels according to the preferences, that is, given a preference $y_i \succ_{\mathbf{x}} y_j$ then $g_\theta(\mathbf{x}, y_i) > g_\theta(\mathbf{x}, y_j)$ should hold, and thus

$$\mathbf{w}^\intercal \psi(\mathbf{x}, y_i) > \mathbf{w}^\intercal \psi(\mathbf{x}, y_j) \Rightarrow \mathbf{w}^\intercal (\psi(\mathbf{x}, y_i) - \psi(\mathbf{x}, y_j)) > 0. \tag{1}$$

Equation (1) can be interpreted as the margin (or *confidence*) of the preference $y_i \succ_{\mathbf{x}} y_j$ and, intuitively, large margin on training instances lead to good generalization capability of the ranker [12].

The instance-label joint representation used in PRL is based on the Kessler's construction for multi-class classification [13, 14, 15, 7]. The Kessler's construction is a very powerful tool to reduce learning problems [6] that allows, through an appropriate instances' representation, to solve multi-class problems using a single linear function instead of decomposing them into many binary sub-problems. The Kessler's construction can be formalized as in the following: given an instance (possibly) embedded in a feature space, i.e., $\phi(\mathbf{x})$, associated with label y, we define the instance-label representation ψ as

$$\psi(\phi(\mathbf{x}), y) = \mathbf{e}_y^m \otimes \phi(\mathbf{x}) = (\ \underset{\uparrow 1}{\mathbf{0}}\ ;\ \underset{\uparrow 2}{\mathbf{0}}\ ; \ldots ;\ \underset{\uparrow y}{\phi(\mathbf{x})}\ ; \mathbf{0}; \ldots ;\ \underset{\uparrow m}{\mathbf{0}}\ ) \in \mathbb{R}^{d \cdot m},$$

where the symbol $\otimes$ indicates the Kronecker product, $\mathbf{e}_y^m$ is the y-*th* canonical basis of

$\mathbb{R}^m$, and $\mathbf{0}$ are d-dimensional zero vectors. Therefore, given a preference $y_i \succ_{\mathbf{x}} y_j$ we can construct its corresponding embedding $\mathbf{z} \in \mathbb{R}^{d \cdot m}$ as

$$\begin{aligned}\mathbf{z} &= \psi(\phi(\mathbf{x}), y_i) - \psi(\phi(\mathbf{x}), y_j) = (\mathbf{e}^m_{y_i} - \mathbf{e}^m_{y_j}) \otimes \phi(\mathbf{x}) \\ &= (\mathbf{0}; \ldots; \underset{\substack{\uparrow \\ y_i}}{\phi(\mathbf{x})}; \mathbf{0}; \ldots; \underset{\substack{\uparrow \\ y_j}}{-\phi(\mathbf{x})}; \mathbf{0}; \ldots; \mathbf{0}) \in \mathbb{R}^{d \cdot m}.\end{aligned}$$

With this definition of a preference $\mathbf{z}$, we can rewrite the margin (Equation (1)) as $\mathbf{w}^\intercal \mathbf{z}$. Note that, if $\mathbf{x}$ is defined directly in the input space, ϕ corresponds to the identity function. At prediction time, given a new instance $\mathbf{x}_{\text{new}}$, labels are ranked according to the score $g_{\mathbf{w}}(\phi(\mathbf{x}_{\text{new}}), y), \forall y \in \mathcal{Y}$. In case of classification, the predicted label for $\mathbf{x}_{\text{new}}$ is the one that maximizes the achieved score, that is,

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} g_{\mathbf{w}}(\phi(\mathbf{x}_{\text{new}}), y).$$

2.2. Game Theory

Game theory is a branch of mathematics that studies the behaviour of rational game players who are trying to maximize their utility. For the purposes of this work, we focus on two-players zero-sum games, which are by definition non-cooperative games.

The strategic form of a two-players zero-sum game is defined by a triplet (P, Q, M), where P and Q are finite non-empty set of (pure) strategies for player P and Q, respectively, and $M : P \times Q \to \mathbb{R}$ is a function that associates a value $M(i, j)$ to each pair of pure strategies (i, j) s.t. $i \in P$, and $j \in Q$. Since P and Q are finite sets, the function M can be represented as a matrix $\mathbf{M} \in \mathbb{R}^{|P| \times |Q|}$, dubbed payoff matrix (or game matrix), such that $\mathbf{M}_{i,j} = M(i, j)$, where $|P|$ and $|Q|$ are the number of available pure strategies for P and Q, respectively. Each matrix entry $\mathbf{M}_{i,j}$ represents the loss of P, or equivalently the payoff of Q, when the strategies i and j are simultaneously played by the two-players.

The game is held in rounds. At each round, the row player P and the column player Q, play simultaneously: P picks a row, while Q picks a column of $\mathbf{M} \in \mathbb{R}^{|P| \times |Q|}$. The correspondent entry in $\mathbf{M}$ is the loss incurred by P or equivalently the payoff of

Q. Clearly, the two players have opposite goals: player P wants to find a strategy that minimizes its loss, while player Q aims at defining a strategy that maximizes its payoff. Typically, the players strategies are randomized over the rows/columns of the game matrix, that is, player P selects a row according to a probability distribution $\mathbf{p}$ over the rows, and, similarly, player Q selects a column according to a probability distribution $\mathbf{q}$ over the columns. This type of strategies are called mixed strategies, and they are typically represented as stochastic vectors, i.e., $\mathbf{p} \in \mathscr{S}_P$ and $\mathbf{q} \in \mathscr{S}_Q$, respectively, where $\mathscr{S}_P = \{\mathbf{p} \in \mathbb{R}_+^{|P|} \mid \|\mathbf{p}\|_1 = 1\}$ and $\mathscr{S}_Q = \{\mathbf{q} \in \mathbb{R}_+^{|Q|} \mid \|\mathbf{q}\|_1 = 1\}$.

It is well known [16] that the best pair of optimal strategies $(\mathbf{p}^*, \mathbf{q}^*)$, that is, the saddle-point (or Nash equilibrium) of $\mathbf{M}$, can be computed by

$$V^* = \mathbf{p}^{*\intercal}\mathbf{M}\mathbf{q}^* = \min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^\intercal \mathbf{M}\mathbf{q} = \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^\intercal \mathbf{M}\mathbf{q}, \tag{2}$$

where V^* is known as the value of the game.

The saddle-point solution of Equation (2) can be found in polynomial time using linear programming.

2.3. Approximating the solution

From a computational point of view, solving high dimensional game matrices through linear programming can become prohibitive. A possible way for addressing this computational issue is to rely on approximated solutions. There is a large body of research in the game theory community which deals with the problem of approximating the value of the game for huge game matrices.

Freund et al. [17, 18] proposed an adaptive approach to compute an approximate saddle-point strategy using multiplicative weights. This algorithm, dubbed *Adaptive multiplicative weights* (AMW), is guaranteed to come close to the minimum loss achievable by any fixed strategy. An incremental version of AMW, called i-AMW, has been recently proposed by Bopardikar et al. [19]. Same authors, previously presented a randomized approach in which each player chooses its best mixed strategy on a sampled set of rows/columns, that is, the payoff matrix is a submatrix of the whole payoff matrix. Authors showed that, with sufficiently large submatrices, there exists a proba-

bilistic guarantees about the quality of the approximation.

In this work we rely on the *Fictitious Play* (FP) algorithm [20] (a.k.a. Brown-Robinson learning process) that is one of the first methods proposed in the literature for approximating the solution of a game. We opt for FP because of its simplicity and its efficiency w.r.t. other approaches like AMW.

FP is a greedy approach that works as follows: a player picks an initial random pure strategy, then, in turn, each player picks its next pure strategy as the best response, assuming the opponent picks at random according to the distribution defined by its previous choices. In other words, at each round both players try to infer the opponent mixed strategy on the basis of its previous selections. The pseudo-code of *FictPlay* is reported in Algorithm 1.

Algorithm 1: FictPlay: Fictitious Play algorithm

Input: $\mathbf{M} \in \mathbb{R}^{P \times Q}$: matrix game,
T_e: number of iterations
Output: $\mathbf{p}, \mathbf{q}$: row/column player strategy,
V: the value of the game

$r \leftarrow randint[1, P]$
$\mathbf{s}_p, \mathbf{v}_p \leftarrow \mathbf{0}, \mathbf{0}$
$\mathbf{s}_q, \mathbf{v}_q \leftarrow \mathbf{M}_{r,:}, \mathbf{e}_r^P$
for $t \leftarrow 1$ **to** T_e **do**
  $\hat{q} \leftarrow \arg\max \mathbf{s}_q, \quad \mathbf{s}_p \leftarrow \mathbf{s}_p + \mathbf{M}_{:,\hat{q}}$
  $\hat{p} \leftarrow \arg\min \mathbf{s}_p, \quad \mathbf{s}_q \leftarrow \mathbf{s}_q + \mathbf{M}_{\hat{p},:}$
  $\mathbf{v}_q \leftarrow \mathbf{v}_q + \mathbf{e}_{\hat{q}}^Q, \quad \mathbf{v}_p \leftarrow \mathbf{v}_p + \mathbf{e}_{\hat{p}}^P$
end
$\mathbf{p} \leftarrow \mathbf{v}_p / \|\mathbf{v}_p\|_1$
$\mathbf{q} \leftarrow \mathbf{v}_q / \|\mathbf{v}_q\|_1$
$V \leftarrow \mathbf{p}^\intercal \mathbf{M} \mathbf{q}$
return $\mathbf{p}, \mathbf{q}, V$

In Algorithm 1, $\mathbf{s}_p$ represents the unnormalized expected value when player Q plays according to $\mathbf{s}_q$. Analogous considerations are valid for $\mathbf{s}_q$. $\mathbf{M}_{r,:}$ and $\mathbf{M}_{:,c}$ indicate the r-th row and the c-th column of the matrix $\mathbf{M}$, respectively. Observe that, given the starting pure strategy for player P, Fictitious Play is deterministic: subsequent executions of *FictPlay* will produce the same strategies $\mathbf{p}$ and $\mathbf{q}$.

Considering that the result of Fictitious Play is an approximation of the optimal

strategies, it is necessary to define bounds to describe the quality of the approximation. In particular, given $\mathbf{q}_t$ and $\mathbf{p}_t$ the approximated strategies after t iterations of Fictitious Play, the bounds are computed as

$$\underline{V} = \min \mathbf{M}\mathbf{q}_t \text{ and } \overline{V} = \max \mathbf{p}_t^\intercal \mathbf{M}.$$

Namely, the lower bound $\underline{V}$ corresponds to the minimum payoff that player P would receive playing the best pure strategy against the mixed strategy $\mathbf{q}_t$ for player Q. On the other hand, the upper bound $\overline{V}$ is the payoff achieved by Q, playing its best pure strategy against the approximated mixed strategy $\mathbf{p}_t$. Subsequent iterations of Fictitious Play will lead (non monotonically) to a better approximation of $\mathbf{q}^*$, and thus to higher lower bounds until convergence is reached.

If the optimal strategy $\mathbf{q}^*$ is found, all pure strategies for P that have weight in the optimal strategy $\mathbf{p}^*$, will lead to the very same upper bound. The same reasoning holds for player Q. Thus, as the Nash equilibrium requires, no player would benefit from changing unilaterally their own strategy. In such case, upper and lower bound are equal to the value of the game and convergence is reached.

3. Parallel Fictitious Play

The computational cost of approximating the optimal strategies for players P and Q using Fictitious Play is $\mathcal{O}(T_e \cdot \max(|P|, |Q|))$. Being actions chosen at time t dependent on $\mathbf{s}_p$ and $\mathbf{s}_q$, which are computed using previously selected pure strategies, Fictitious Play cannot be directly parallelized.

Empirically, the quality of the solution, computed as the gap between bounds (see Figure 1(a)), has a fast initial drop toward the optimal strategies and a subsequent slow asymptotic convergence toward the real values.

To exploit the advantages of the first part of the search, without burdening the algorithm with the second part (slow and less fruitful), we propose a simple, yet effective, parallel research of the optimal strategies. This version of *FictPlay*, which will be referred to as *Parallel FictPlay*, consists in computing (possibly in a parallel fashion)

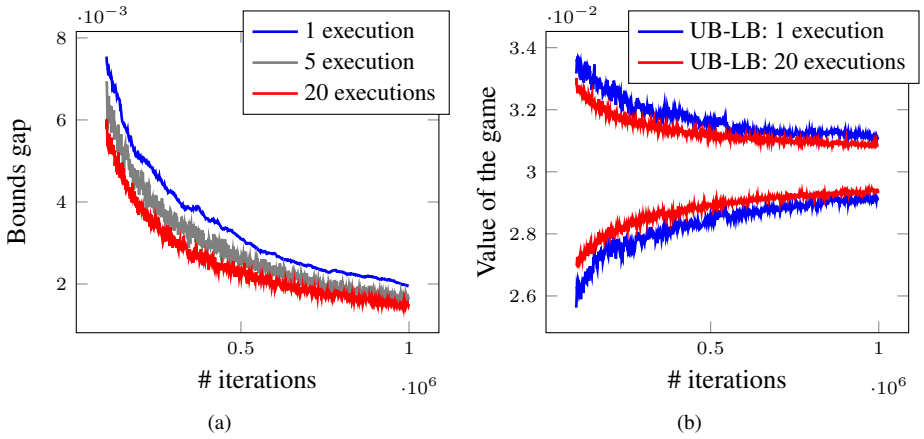


Figure 1: (a) Distance between lower and upper bound on the game value either using mixed strategies approximated by a single run of Fictitious Play algorithm or averaging over multiple executions. The payoff matrix$\in \mathbb{R}^{958\times(27^2\cdot 958)}$ is based on a polynomial version in a preference learning setting of `tic-tac-toe` dataset and describes a zero sum game, where the players can either win (+2), lose (-2) or have a tie (0). The value of the game is approximatively 0.3. (b) Lower and upper bound on the value of the game described by the same matrix used to build the plot in Figure 1(a).

different approximated strategies $\{\mathbf{p}_1^{(t)}, \ldots, \mathbf{p}_k^{(t)}\}$ and $\{\mathbf{q}_1^{(t)}, \ldots, \mathbf{q}_k^{(t)}\}$, and then averaging over the found strategies. Empirically, Figure 1(a) and Figure 1(b), show how *Parallel FictPlay* can achieve better performances, with more strict bounds on the approximated solutions. Given the low convergence rate as the bounds approach the real value of the game, Parallel FP can achieve similar bounds to FP in almost half of the iterations.

Finally, it can be noted that, given the deterministic nature of Fictitious Play, to obtain different strategies using *parallel FictPlay* it is necessary, yet not sufficient, that the different executions of *FictPlay* have different starting points. Under this assumption, we can see *parallel FictPlay* as the parallelized version of sequential *FictPlay* with random restarts.

Theorem 1. *Parallel FictPlay converges to the optimal solution.*

Proof. We know that a single execution of FP converges to the optimal solution, so there exists a number of iterations $t \geq T$ and an arbitrary small $\epsilon \geq 0$ such that

$$\forall i \in [1, P],\ p_i^* - \epsilon \leq p_i^{(t)} \leq p_i^* + \epsilon \quad \text{and} \quad \forall i \in [1, Q],\ q_i^* - \epsilon \leq q_i^{(t)} \leq q_i^* + \epsilon.$$

Given an execution of *Parallel FP* with k FP, its strategy is the average of the strategies of the single FP. Thus, given T and ϵ we have that:

$$\overline{\overline{\mathbf{p}}}^{(t)} = \frac{1}{k}\sum_{s=1}^{k}\mathbf{p}_s^{(t)}, \quad \overline{\overline{\mathbf{q}}}^{(t)} = \frac{1}{k}\sum_{s=1}^{k}\mathbf{q}_s^{(t)}$$

where $\overline{\overline{\mathbf{p}}}^{(t)}$ and $\overline{\overline{\mathbf{q}}}^{(t)}$ are the strategies of *Parallel FP* after T iterations.

Thus, we can bound the difference of $\overline{\overline{\mathbf{p}}}^{(t)}$ and $\overline{\overline{\mathbf{q}}}^{(t)}$ w.r.t. the optimal strategies, i.e.,

$$\forall i \in [1, P],\ p_i^* - \epsilon \leq \overline{\overline{p}}_i^{(t)} \leq p_i^* + \epsilon \quad \text{and} \quad \forall i \in [1, Q],\ q_i^* - \epsilon \leq \overline{\overline{q}}_i^{(t)} \leq q_i^* + \epsilon.$$

Since ϵ is a bound for all the single FP, we can also affirm that on expectation the *Parallel FP* bound on the strategies' entries is stricter than ϵ. This is easy to show since the bound is ϵ also for *Parallel FP* iff for all $s \in [1, k]$ there exists an entry in either $\mathbf{p}_s^{(t)}$ or $\mathbf{q}_s^{(t)}$ such that it differs from the corresponding entry in $\mathbf{p}^*$ or $\mathbf{q}^*$ of exactly $\pm\epsilon$. In all other cases, averaging over the strategies guarantees a stricter bound than ϵ.

□

4. Preference Learning: a game theoretic perspective

In this section we introduce the theoretical principles that underlies PRL. Throughout the section we assume a training set of N preferences of the form $(y_+ \succ_{\mathbf{x}} y_-)$. Such preferences are converted into their corresponding vectorial representation using the Kessler's construction as described in Section 2.1.

As mentioned previously, we consider an hypothesis space $\mathcal{H}$ composed by linear functions of preference representations, i.e., $\mathcal{H} \equiv \{\mathbf{z} \mapsto \mathbf{w}^\intercal\mathbf{z} \mid \mathbf{w}, \mathbf{z} \in \mathbb{R}^{d\cdot m}\}$. Given a preference $\mathbf{z}$, we say that $\mathbf{z}$ is satisfied by a hypothesis $\mathbf{w}$ iff $\mathbf{w}^\intercal\mathbf{z} > 0$, that is, when the margin of the preference $\rho(\mathbf{z}) = \mathbf{w}^\intercal\mathbf{z}$ is strictly positive. The margin of a preference can be considered as the confidence of the hypothesis $\mathbf{w}$ over the preference $\mathbf{z}$.

PRL aims at finding the linear hypothesis $\mathbf{w}$ in the preference space that maximizes the minimum margin over the training preferences. According to the Representer Theorem [21, 22] we know that the maximal margin hypothesis $\mathbf{w}$ can be defined as a

convex combination of the training preferences, that is

$$\mathbf{w} \propto \sum_j \alpha_j \mathbf{z}_j, \boldsymbol{\alpha} \in \mathscr{S}_P.$$

Hence, the margin of a preference $\mathbf{z}$ can be rewritten as

$$\begin{aligned}\rho(\mathbf{z}) = \mathbf{w}^\intercal \mathbf{z} = \sum_{j=1}^{N} \alpha_j \mathbf{z}_j^\intercal \mathbf{z} = \sum_{j=1}^{N} \alpha_j \sum_{f \in \mathcal{F}} \mu_f \mathbf{z}_j[f]^\intercal \mathbf{z}[f] \\ = \sum_{j=1}^{N} \sum_{f \in \mathcal{F}} \alpha_j \mu_f \mathbf{z}_j[f]^\intercal \mathbf{z}[f] = \sum_{(j,f)} q_{(j,f)} \mathbf{z}_j[f]^\intercal \mathbf{z}[f],\end{aligned}$$

where the dot product $\mathbf{z}_j^\intercal \mathbf{z}$ is generalized by assigning weights to the features according to a distribution $\boldsymbol{\mu}$ over the features, and $\mathbf{q}$ is a new distribution over all the possible preference-feature pairs such that $q_{(j,f)} = \alpha_j \mu_f$. $\mathcal{F}$ is the (potentially infinite) set of (possibly non-linear) features and $\mathbf{z}[f]$ is the sub-vector of $\mathbf{z}$ corresponding to the f-th feature in the Kessler's construction.

Now, let assume that an adversary chooses a distribution $\mathbf{p}$ over the training preferences with the goal of minimizing the expected margin achieved by the hypothesis $\mathbf{w}$ on the training preferences. Given $\mathbf{p}$, the expected margin will be defined by

$$\bar{\rho}(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{N} p_i \sum_{(j,f)} q_{(j,f)} \mathbf{z}_i[f]^\intercal \mathbf{z}_j[f] = \mathbf{p}^\intercal \mathbf{M} \mathbf{q} \tag{3}$$

where $\mathbf{M}_{i,\pi(j,f)} = \mathbf{z}_i[f]^\intercal \mathbf{z}_j[f]$, with π the function which maps preference-feature pairs onto univocal indexes, i.e., $\pi_{(j,f)} \in [1, N|\mathcal{F}|]$.

It is pretty evident that Eq. (3) has a strong relation with the two-players zero-sum game presented in Section 2.2. Specifically, consider a two-players zero-sum game where the row player P (the nature) picks a preference from a distribution over the whole set of training preferences (i.e., the rows) aiming at minimizing the expected margin $\bar{\rho}$. Simultaneously, the opponent player Q (the learner) picks a column from a distribution over the set of preference-feature pairs (i.e., the columns) aiming at maximizing the expected margin (payoff). Then, the value of the game, that is the maximal

minimum margin solution is given by

$$V = \bar{\rho}(\mathbf{p}^*, \mathbf{q}^*) = \min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^\intercal \mathbf{M} \mathbf{q},$$

that is exactly the same as Equation (2). In other words, searching for the distribution $\mathbf{q}$ maximizing the minimum margin in the training set is equivalent to find the saddle-point solution of the game matrix $\mathbf{M}$.

5. PRL: Preference and Rule Learning

The number of columns of the game matrix $\mathbf{M}$ is equal to the number of all possible preference-feature pairs., i.e., $N|\mathcal{F}|$. In general, such amount is huge and solving the game using standard off-the-shelf algorithms from game theory is infeasible. Unfortunately, using approximated methods does not solve the issue especially because, potentially, the number of columns can also be infinite ($|\mathcal{F}| \to \infty$).

To overcome this problem, we propose a new incremental method for solving the game, that we call PRL (Preference and Rule Learning algorithm). The main idea behind PRL is to consider only a fraction of the columns of the whole game matrix. Iteratively, each sub-game is solved and the columns that do not contribute to the strategy of the column player are replaced by new randomly selected columns.

Formally, let $\mathbf{M}$ be the game matrix and let $(\mathbf{p}^*, \mathbf{q}^*, V^*)$ be its corresponding optimal solution. At each iteration the algorithm considers a subset of columns of $\mathbf{M}$, that is $\mathbf{M}_t = \mathbf{M}\mathbf{\Pi}_t$ where $\mathbf{\Pi}_t \in \{0,1\}^{Q \times B}$ are left-stochastic (0,1)-matrices, i.e., matrices whose entries belong to the set $\{0, 1\}$ and whose columns add up to one. B, that we call budget, is the number of columns considered at each iteration.

Let now consider the solution $(\mathbf{p}_t^*, \mathbf{q}_t^*, V_t^*)$ of the matrix $\mathbf{M}_t$ computed at iteration t. At the end of each iteration, the algorithm replaces the columns of $\mathbf{M}_t$ corresponding to null entries in $\mathbf{q}_t^*$ (which do not contribute in the solution) with new columns randomly drawn from the whole set of available columns. In this setting, the following theorem holds.

Theorem 2. *In PRL, at each iteration, the value of the game increases monotonically*

and it is upper bounded by the optimal margin, that is the value of the game when considering the full matrix $\mathbf{M}$.

Proof. Let assume of being at iteration $t+1$: a new left-stochastic (0,1)-matrix $\mathbf{\Pi}_{t+1}$ is taken into account that is $\mathbf{\Pi}_t$ where every row corresponding to null entries in $\mathbf{q}_t^*$ have been substituted with a new random stochastic vector $\mathbf{e}_h^Q$. Thus, it holds that

$$
\begin{aligned}
V_t^* = \mathbf{p}_t^{*\intercal}\mathbf{M}_t\mathbf{q}_t^* &= \mathbf{p}_t^{*\intercal}\mathbf{M}\mathbf{\Pi}_t\mathbf{q}_t^* && (4)\\
&\leq \mathbf{p}_{t+1}^{*\intercal}\mathbf{M}\mathbf{\Pi}_t\mathbf{q}_t^* && (5)\\
&= \mathbf{p}_{t+1}^{*\intercal}\mathbf{M}\mathbf{\Pi}_{t+1}\mathbf{q}_t^* && (6)\\
&\leq \mathbf{p}_{t+1}^{*}\mathbf{M}\mathbf{\Pi}_{t+1}\mathbf{q}_{t+1}^* && (7)\\
&= \mathbf{p}_{t+1}^{*\intercal}\mathbf{M}_{t+1}\mathbf{q}_{t+1}^* = V_{t+1}^* && (8)
\end{aligned}
$$

and

$$
\forall t, V_t^* = \mathbf{p}_t^{*\intercal}\mathbf{M}\mathbf{\Pi}_t\mathbf{q}_t^* \leq \mathbf{p}^{*\intercal}\mathbf{M}\underbrace{\mathbf{\Pi}_t\mathbf{q}_t^*}_{\tilde{\mathbf{q}}_t} \leq \mathbf{p}^{*\intercal}\mathbf{M}\mathbf{q}^* = V^*.
$$

Equivalence (4) is trivial since $\mathbf{M}_t = \mathbf{M}\mathbf{\Pi}_t$ by definition. Inequality (5) holds because the strategy $\mathbf{p}_{t+1}^*$ is suboptimal for $\mathbf{M}_t$. In (6) we simply replaced columns of the game matrix corresponding to null entries of $\mathbf{q}_t^*$ which does not affect the value. Finally, inequality (7) is true because $\mathbf{q}_t^*$ is suboptimal for $\mathbf{M}_{t+1}$, and similar considerations can be done for the last series of inequalities. □

The pseudo-code of the full algorithm is given in Algorithm 2.

It is worth to notice that the algorithm does not require any prior knowledge about $\mathbf{M}$, and the number of columns can be also infinite. Thus, a natural approach for dealing with potentially infinite game matrices is to use an on-line column generation approach as we will discuss in Section 6.

Figure 2 shows a visual overview of PRL. In the figure the three main phases of PRL are highlighted: (i) the learning phase takes the training preferences and the feature generator to produce the game matrix that it is incrementally solved as described in Section 5; (ii) the learned hypothesis is then used to make prediction for unseen

Algorithm 2: PRL: Preference and Rule Learning

Input: $\mathcal{P}$: set of training preferences
F_{gen} : random feature generator
B: size of the working set
T: number of epochs
T_e: number of iterations of *FictPlay*

Output: $\mathcal{Q}$: working set of hypothesis
$\mathbf{q}$: mixed strategy in $\mathcal{Q}$

random initialization of the set $\mathcal{Q}$ such that $|\mathcal{Q}| = B$
compute the matrix game $\mathbf{M}$ on the basis of $\mathcal{P}$ (rows) and $\mathcal{Q}$ (cols)
for $t \leftarrow 1$ **to** T **do**
    $\mathbf{p}, \mathbf{q}, v \leftarrow$ *FictPlay*$(\mathbf{M}, T_e)$
    if $t < T$ **then**
        foreach $(j, f) \mid \mathbf{q}_{(j,f)} = 0$ **do**
            $(j', f') \leftarrow$ *pick*$(\mathcal{P}), F_{gen}()$
            update $\mathcal{Q}$: replace (j, f) with (j', f')
            update columns of $\mathbf{M}$ w.r.t. $\mathcal{Q}$:
                let k the position of (j', f') in $\mathcal{Q}$,
                for all $i \in \mathcal{P}$, $\mathbf{M}_{i,k} = \mathbf{z}_i[f]^\intercal \mathbf{z}_{j'}[f]$
        end
    end
end
return $\mathbf{q}$, $\mathcal{Q}$

preferences, and when it is possible (iii) the prediction is explained using the learned feature weights.

5.1. PRL with dynamic budget

Generally speaking, there is not a valid heuristic for setting the budget B to a value that can give some guarantees. For this reason the hyper-parameter B could be over-estimated leading to poor efficiency of the algorithm, or conversely, underestimated leading to weak solutions. To tackle this problem, we propose a variant of PRL with dynamic budget. The main difference w.r.t. the classical PRL is that B now represents the minimum number of columns that are replaced at each iteration, and not the total number of available columns.

Let us make an example. Let $B = 10$ and let the initial number of columns of $\mathbf{M}$ be equal to 20. Let assume that after iteration 1, there are 14 columns such that

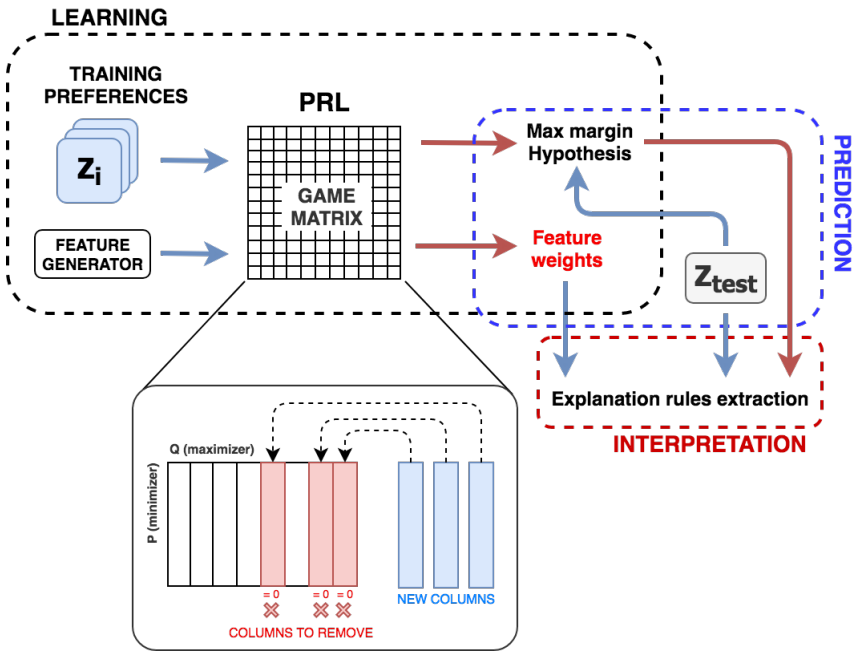


Figure 2: Visual overview of PRL. The black section represent the training phase with a "zoom" on the columns replacement policy; The blue section is the prediction phase that is necessary for the interpretation phase (red section) which uses the learned feature weights and the test preference to interpret the prediction.

$\mathbf{q}_{(j,f)} > 0$. Then, at the end of iteration 1, these 14 columns will be kept in $\mathbf{M}$ and other B randomly generated columns will be added (actually, 6 columns are replaced and 4 added). In this way, the total number of columns of $\mathbf{M}$ at iteration 2 will be 24.

The drawback of this technique is that the number of column of $\mathbf{M}$ can potentially become very big. However, empirically (see Section 7), in all the performed experiments, $\mathbf{M}$ has always kept a reasonable number of columns (not much higher than the initial B). It is worth noticing that for this variant of the algorithm all the theoretical properties of PRL are preserved.

6. On-line feature generation

In PRL, on-line columns generation is one of the most important component. As mentioned in the previous sections each column of the game matrix is defined by a preference-feature pair. So, in order to generate a column, we need to independently

draw a preference and a feature. Hence, it is crucial to define how the on-line feature generator works. We propose different feature generation schemes based on different types of features, specifically: polynomial features, decision rules, and decision tree paths.

In Algorithm 2 the function F_{gen} refers to a generic feature generator scheme.

6.1. Polynomial features generation

The polynomial feature generation scheme produces features that are taken from the feature space of the homogenoeus polynomial kernel. For instance, given an n-dimensional instance $\mathbf{x}$ some possible polynomial features of degree 3 are: $x_1x_2x_n$, $x_1^2x_3$ and x_n^3. It is worth to notice that, when the input variables are binary-valued, polynomial features are highly interpretable since monomials correspond to logical conjunctions, e.g., if $x_i \in \{0, 1\}$ then $x_1x_2x_n := x_1 \wedge x_2 \wedge x_n$.

6.2. Rules generation

When it comes to interpret machine learning models, logical rules (as in decision trees) are the most natural choice. In order to introduce interpretable features in PRL, we propose a rules generator scheme. To generate rules, we must take into account the nature of the input variables. For example, in the case of binary valued input variables a rule is simply their truth value. However, when dealing with continuous variables a rule can be defined as a relation involving the values of the variables, e.g., by defining a threshold like $x \geq 5$, or by checking the exact value such as $x = 3.2$. For generating these type of rules, the generator randomly picks a continuous feature f, and a random threshold value taken from the set of values assumed by f in the training set. Finally, a random relation is drawn from the set $\{\leq, \geq, =\}$. In some of the experiments we used a reduced set of relations (this is specified in Section 7). Note that discrete variables can be considered as a special case of binary variables since it is possible to convert them into binary ones through one-hot encoding.

Finally, the generated rules can be also combined using conjunctions. Specifically, given two or more rules, their product corresponds, from a logical point of view, to the conjunction of the conditions. In the remainder we will refer to the arity of this combination as the degree of the rule.

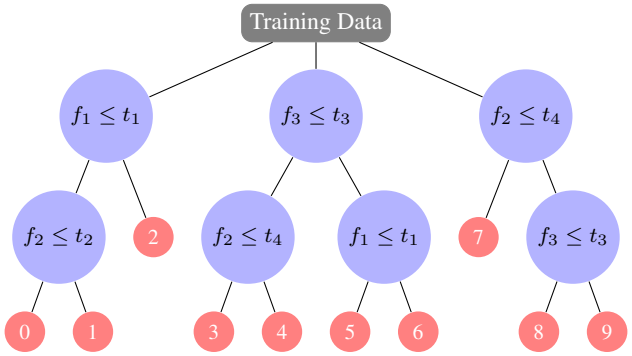


Figure 3: Simple example of a random forest with three decision trees: light blue circles are the decision nodes, while the small red circles are the end nodes. In the figure it is assumed that left branches mean that the rules in the decision nodes are satisfied. Each leaf represents a possible decision rule that the feature generator can extract. In particular, the depicted forest allows to extract 10 possible rules. However, there is a pair of rules that are actually the same: ④=⑧, i.e., $(f_2 > t_4) \wedge (f_3 \leq t_3)$.

6.3. Decision path rules generation

Generating rules as described in Section 6.2 can not be optimal since the rules are generated in a completely random fashion. The decision path rules generator tries to overcome the limitation of the previous method by taking advantage of the nature of the decision paths in decision trees (DTs). DT paths are based on a split criterion that is usually defined in terms of some entropic index.

At classification time a DT takes the instance and traverses the tree according to the value of the features in the split nodes. When an example reaches a leaf, it means that it has satisfied all the rules along the decision path. The Decision path rule generator, given a random forest, picks at random one of the possible decision path (i.e., conjunction of relations) from a randomly picked tree of the random forest.

Figure 3 shows an example of a random forest composed by three decision trees. The total number of decision nodes corresponds to the total number of possible decision paths. However, some paths represent the same rule, e.g., ④=⑧, i.e., $(f_2 > t_4) \wedge (f_3 \leq t_3)$. In PRL, once the set of all possible decision paths is extracted, the Decision path rule generator randomly picks one path and generates the corresponding rule.

6.4. Bias feature

As we will see in the experimental section, whenever a class can be defined by reasonably simple rules, PRL has the capability of identifying them all. However, when there are classes which cannot be characterized by rules (or the rules are too complex) PRL may fail in extracting reasonable explanations. This is due to the fact that when a class A is logically defined as the negation of another class B, by design PRL still searches for features that characterize $A \succ B$ even though it cannot be defined in a reasonable amount of rules.

Let us make a toy example using the tic-tac-toe game to explain this concept. The task is to classify whether a tic-tac-toe configuration is a win for the cross ($\times$) or not. It is clear that it is quite simple to define when there is a win: whenever there are three crosses in line. However, how can a configuration that is not a win be defined? The easiest way is by rejecting all the winning configurations. Otherwise the only way is analytically describing each non winning configuration for the cross that is however not convenient. Unfortunately, in this scenario the best PRL can achieve is to identify a set of rules that are able to discriminate only small subsets of not winning training instances, but with rather small generalization capabilities. In some sense we can say that PRL is overfitting.

To address this issue, we introduce an artificial feature (i.e., rule) that is set to be true for every preference. That is, each example in the training set has the feature $\top$ whose value is equal to 1. Then, we allow the feature generation mechanism to pick the $\top$ rule together with the other rules. We will call it *bias* rule/feature. When such feature is selected and associated with a preference, it will give a bias towards the preferred class. Clearly, the bias feature per se has no discriminative capabilities, thus if a class can be characterize with a small subset of rules PRL will still be able to do it. However, in cases similar to the tic-tac-toe example, where a class is simply the negation of the other, then the bias rule will play a key role. All examples of such class trivially satisfy the bias rule and hence its weight will be reasonably high. This can be interpreted as "label A is preferred to label B because there is no evidence to say the opposite".

The bias feature has been used in all experiments concerning the `poker` dataset and also in some experiments on `tic-tac-toe`.

7. Evaluation

In this section, we describe the empirical evaluation of PRL[1] and the assessment of its effectiveness. Specifically, our experiments focused on three main aspects associated with PRL: evaluating the degree of interpretability of the proposed model, the possibility of visualizing the model decisions, and assessing the performance of the feature selection.

In all the experiments the number of iterations T_e of *Parallel FictPlay* has been set to 10^5 (8 parallel executions of *FictPlay* have been run), while the number T of epochs of PRL has been set to 10^3. All experiments have been performed using dynamic budget PRL with initial budget $B = 500$. We group the set of experiments on the basis of their purposes. The first set aims to assess the degree of interpretability of PRL as well as its effectiveness on some benchmark datasets. The second set of experiments, instead, focuses on the evaluation of the performance on datasets with a huge number of features.

7.1. Model interpretation

In the first set of experiments, we employed PRL to select the most relevant features for interpreting the decisions. We ran PRL on four benchmark datasets. The details of the datasets are summarized in Table 1.

Dataset	**#Instances**	**#Features**	**#Classes**
`tic-tac-toe`	958	27	2
`breast-cancer`	682	9	2
`poker`	25010	52/69/74*	10**
`mnist`	10000	784	10

Table 1: Datasets information: name, number of instance, number of features, and number of classes. All the dataset are freely available in the UCI repository. (*) the `poker` dataset has 3 versions with different number of features. (**) the original dataset has 10 classes, however in our experiments three binary classification tasks have been defined.

[1] Implementation available at `https://github.com/makgyver/PRL`

7.1.1. The `tic-tac-toe` *dataset*

As a general test bed, we selected the `tic-tac-toe` dataset, in which each example describes a possible final configuration of the tic-tac-toe game, and examples are labeled as positive iff the $\times$ player is the winner. The dataset has been converted into a binary-valued dataset through one-hot encoding, obtaining 27 binary input variables for each instance. The 27 features represent a specific position on the board, in which each cell can have a *cross* ($\times$), a *nought* ($\circ$) or can be *empty*. Each cell is encoded in three consecutive binary features x_i, x_{i+1}, x_{i+2} where x_i means *empty*, x_{i+1} means *nought* and x_{i+2} means *cross*, for $i = 3n$ with $n \in [0, \dots, 8]$. Note that with this encoding the positive class can always be expressed as a single DNF rule which describes all the possible eight 3-cross-in-a-line configurations, and negative otherwise.

A winning position is characterized by the simultaneous activation of three specific features (either columns, diagonals or rows), thus can use such a-priori knowledge to select an appropriate feature space. For example, polynomial features are suited for this purpose because they correspond to conjunctions when the input features are binary. Thus, we used the polynomial features generator of degree 3. The experiment setup was the following: 70% of the dataset has been used as training set, while the remaining 30% was used as test set.

After the training phase, the top 10 features that PRL weighted the most were the following: $x_8x_{17}x_{26}$, $x_2x_{11}x_{20}$, $x_2x_{14}x_{26}$, $x_8x_{14}x_{20}$, $x_{20}x_{23}x_{26}$, $x_{11}x_{14}x_{17}$, $x_2x_5x_8$, $x_5x_{14}x_{23}$, $\tilde{x}^3_{13}$ and $\tilde{x}^3_{25}$. Features marked with $\sim$ are the ones characterizing a negative preference (no win for $\times$). Observe how the first 8 features correspond correctly to the winning configurations (three-in-a-line) for the cross. The remaining features represent respectively a naught in the central and in the bottom right cell. The last one does not seem to be particularly informative. Conversely, the former suggests negative evidence that cross won, since occupying the centre is often a good strategy. In fact, a naught in the centre is correctly associated with a negative preference (if naught occupies the centre, it is less likely that cross has won). This highlights the strong interpretability of the model. Overall, the algorithm has been able to identify all the conditions that determine a win for the cross.

Moreover, using the `tic-tac-toe` dataset, we provide a further investigation on the bias feature: we ran two different versions of PRL with degree 3 polynomial features generator: one including the bias feature and one without it. Figure 4 shows the weights associated with rules. Both versions correctly identify as strongest rules the three-in-a-line for the cross (first 8 rules). As 9th feature, we found either the bias feature (in the version of PRL that includes it) or, as previously described, a naught in the centre of the grid. This is particularly evident considering the plots in Figure 4. Without the bias, negative rules, that can describe a losing configuration for cross, are more and more relevant (higher tail for the red line). When the bias feature is available, the weight shared among rules for non-winning configurations is aggregated onto the bias feature. Additionally, rules that describe winning situations are weighted more. The bias feature expresses a single rule that says "if the configuration is not winning for cross, then it is either losing or a tie", without specifying how a non-winning configuration looks like.

Other than using polynomials of degree 3, we tried to extract logical rules as proposed in Section 6.2. Polynomial features correspond to conjunctions of positive literals, while through the rule generation scheme we can encode also negative literals. This allows the introduction of new rules, such as $\neg x_1 \wedge \neg x_{13} \wedge \neg x_{25}$ to which PRL assigned an high weight. In fact, this rule expresses, in a human-readable fashion, the concept that exist some winning configurations with no naught on the diagonal. It is easy to demonstrate that the suggested rule (although not intuitive for a human being) correctly describes a sufficient (yet not necessary) condition to define a winning configuration for cross. If there isn't any naught on the diagonal, then it cannot be a winning configuration for naught. Moreover, it cannot be a tie, since any tie requires the entire grid to be filled, but if no naught is on the diagonal, then the diagonal is occupied by crosses, and thus it must be a winning grid for cross.

7.1.2. The `poker` dataset

The `poker` dataset consists of a set of examples representing poker hands. Each hand is composed by 5 cards taken from a standard poker deck of 52 cards, with 4 suits and 13 ranks (Ace to King) for each suit. The original task of this dataset is to identify

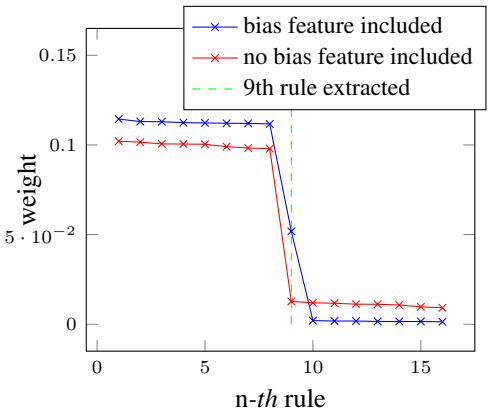


Figure 4: Rule weights of the most relevant rules extracted by PRL on `tic-tac-toe` using polynomial feature generator of degree 3 with (blue) and without (red) the bias feature.

the value of the hand. The 10 possible hand values are: Nothing, Pair, Double pairs, Three of a Kind (TOK), Straight, Flush, Full house, Four of a kind (FOK), Straight flush and Royal flush.

The experiments performed on the `poker` dataset was intended to assess the quality of the features selected by PRL to explain the decision. To this end, we have created a hierarchy of features with the aim of investigating whether PRL could identify, at each level, the best subset of features/rules useful to accomplish the task. The hierarchy of features was defined as in the following:

- **Level 1**: The first representation of the dataset describes a poker hand as a trivial enumeration of the cards contained in it. The hand is described through a vector of dimension 52, where each dimension corresponds to a specific card and has value equal to 1 or 0, whether the card is present in the hand or not: $[\text{A}\heartsuit, 2\heartsuit, \ldots, \text{A}\diamondsuit, 2\diamondsuit, \ldots, \text{J}\spadesuit, \text{Q}\spadesuit, \text{K}\spadesuit] \in \{0, 1\}^{52}$.

- **Level 2**: The second level considers aggregated features obtained by counting either the suits or the ranks of the cards in the hand. Specifically, this new level is obtained by adding 4 new dimensions to the previous 52, that describe the counting of the suits, and 13 additional dimensions that describe the counting of the ranks, i.e., $[\#\heartsuit, \#\diamondsuit, \#\clubsuit, \#\spadesuit] \in [0, 5]^4$ and $[\#\text{A}, \#2, \#3, \ldots, \#\text{Q}, \#\text{K}] \in [0, 4]^{13}$.

Note that the Ace is assumed of rank 1, while J=11, Q=12, and K=13.

- **Level 3**: In this level we create 5 features that are a further aggregation of the features in Level 2. Specifically:
  - $\max\{\#\heartsuit, \ldots, \#\spadesuit\}$: number of cards of the most popular suit in the hand;
  - $\max\{\#\text{A}, \ldots, \#\text{K}\}$: number of cards of the most popular rank in the hand;
  - card$(\{\heartsuit, \ldots, \spadesuit\})$: number of different suits in the hand;
  - card$(\{\text{A}, \ldots, \text{K}\})$: number of different ranks in the hand;
  - $\max(\text{diff}(\text{ranks}))$: the largest ranks difference between two cards in the hand. In this case we associate to the Ace the rank 1 or 14 which minimizes the maximum difference between the other cards.

Let us make an example to clarify this features hierarchy. Given the hand A$\heartsuit$, 7$\heartsuit$, A$\spadesuit$, J$\spadesuit$, 3$\clubsuit$, at the first level of the hierarchy 5 entries out of 52, the ones corresponding to the cards in the hand are equal to 1 and all the rest are zero. At the second level, the suit vector is [2, 0, 1, 2], while the rank vector is [2, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]. These vectors are appended to the previous one giving a 69 dimensional vector. At the third and final level the features values are the following:

- $\max\{\#\heartsuit, \ldots, \#\spadesuit\} = \#\spadesuit = \#\heartsuit = 2$;
- $\max\{\#\text{A}, \ldots, \#\text{K}\} = \#\text{A} = 2$;
- card$(\{\heartsuit, \ldots, \spadesuit\}) = \text{card}(\{\heartsuit, \spadesuit, \clubsuit\}) = 3$;
- card$(\{\text{A}, \ldots, \text{K}\}) = \text{card}(\{\text{A}, 3, 7, \text{J}\}) = 4$;
- $\max(\text{diff}(\text{ranks})) = \text{J} - A = 10$. In this case the Ace is associated with the value 1 since $\text{J} - 1 < 14 - 3$,

which produces the vector $[2, 2, 3, 4, 10]$. Thus, in this last level the total number of features is 74.

We defined three binary classification tasks: TOK (2.05% of the dataset) versus rest, Flush (0.22% of the dataset) versus rest, and Straight (0.37% of the dataset) versus

rest. It is worth noticing that these combinations are included in more valuable hands, for example: TOK is also part of the Full house and the FOK. However, a TOK is not a FOK or a Full house. This fact can cause false positives at classification time.

Alongside evaluating the extracted rules, we assessed the quality of our model using balanced accuracy. Balanced accuracy has been chosen over the more standard accuracy measure due to the strong imbalance between the positive class and the negative one. The balanced accuracy is defined as:

$$\text{BACC} = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right) \times 100,$$

where TP stands for true positives (P = positives), and TN for true negatives (N = negatives).

PRL has been compared against SVM with polynomial and RBF kernels. In particular, SVM has been validated via 5-fold cross validation so that the *C* hyper-parameter was validated among the set $\{10^{-3}, \ldots, 10^{4}\}$, the degree of the polynomial kernel in the range [1,3], and the shape parameter γ of the RBF kernel has been validated in the set $\{10^{-2}, \ldots, 10^{2}, (\#\text{ features} * var(X))^{-1}\}$. PRL has been trained using rules of degree 1 on the set of relations $\{=\}$. A possible example of rule is the following: $x_{13} = 1$ which corresponds to stating that the fourteenth feature, i.e., the ace of diamonds, is set to 1, and thus available in the hand. Experiments have been performed using a 70-30% training and test split division. Moreover, our model has been compared against the Random Forest Classifier, validated using 5-fold cross validation, selecting the hyper-parameter associated with the number of estimators in the set $\{10, 100, 1000, 5000\}$, the maximum depth in $\{2, 5, 10, \text{until pure leaves}\}$ and the split criterion in $\{\textit{Gini}, \textit{entropy}\}$

The achieved results are reported in Table 2. With the name PRL we refer to PRL with the rule generation scheme, while PRL-RF means PRL with the decision path rule generation scheme.

A remarkable difference w.r.t. the SVM can be noticed in the *Straight* classification, which is the hardest task. SVM simply classifies the majority of instances as negative due to the strong unbalancing of the dataset. This behavioural pattern is repeated in

Method	# Level	TOK	Straight	Flush
SVM	1	**79.20**	**53.00**	**59.38**
RF	1	51.76	50.00	50.00
PRL	1	48.08	49.97	59.03
PRL-RF	1	61.42	51.48	50.00
SVM	2	85.42	**56.04**	59.43
RF	2	99.62	50.00	59.38
PRL	2	**100.00**	52.72	**84.36**
PRL-RF	2	98.10	52.72	**84.36**
SVM	3	99.99	96.43	96.85
RF	3	**100.00**	90.90	**100.00**
PRL	3	**100.00**	**100.00**	**100.00**
PRL-RF	3	**100.00**	**100.00**	**100.00**

Table 2: Balanced accuracy (%) on the `poker` dataset. The highest accuracies in all classification tasks, and in all levels, are highlighted in **bold**.

all the tasks (except for TOK) both at the second and at the first level of the hierarchy. Concerning the third level, SVM had one false negative in the *TOK* task and one false positive in the *Flush*, achieving a BACC of 99.99% and 96.43%, respectively.

As shown in Table 2 in the first task, i.e., three of a kind, the best performance is achieved by SVM. We explain this due to the different kind of combination of features considered. Rule-based algorithms like random forest or PRL have to enumerate cards appearing in the three of a kind: on this dataset, where each hand appears only once, this approach is unable to generalize. On the other hand, SVM, in particular the version based on a polynomial kernel of degree 2, tries to describe a three of a kind using couples of cards having the same rank. Due to the fact that a couple of cards of the same rank happens to be in two different versions of a three of a kind (depending on the remaining card of the three), this mechanism is somehow able to generalize to unseen examples and thus achieves better results in previously unseen examples. PRL-RF (the PRL version based on rules extracted from threes of a random forest) achieves the second best result: we guess this to be due to the fact that this approach can consider a number of conjunctions of cards of cardinality 2 (namely the pair of cards) and thus behaves similarly to SVM. The first level of features is unable to explain a Straight hand nor a Flush hand. PRL is completely able to gather rules to explain

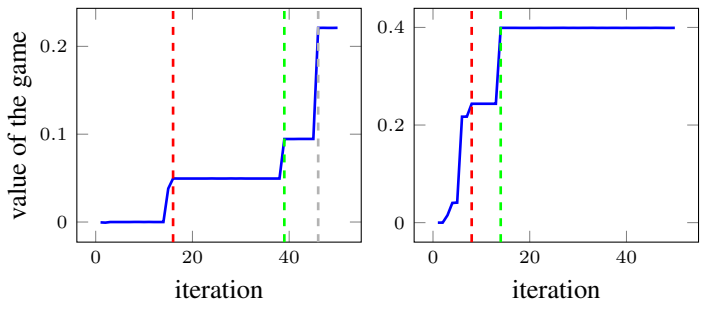


Figure 5: Value of the game w.r.t. the iteration of PRL on the *Straight* (left) and *Flush* (right) task.

a three of a kind using the second level of features, whilst SVM achieves the worst results, probably relying mainly on level 1 features and exploiting only partially level 2 features. Almost all algorithms performs poorly on the straight hand task, using level 2 features. Specifically, an hand contains a TOK anytime one of the rank has a cardinality $= 3$. The rule "cardinality 3 of a rank", yet suffers when it comes to false positives detection: a full house hand is a false positive for such rule. Similarly, a flush can be described with a suit of cardinality 5, but this also includes the straight flush and the royal flush as false positives. In both these tasks, PRL found the correct rules.

With the features contained in the third and final level of the hierarchy it is possible to define all the considered concepts:

- *TOK*: # of ranks = 3, # of cards of the most popular rank = 3;
- *Straight*: # of ranks = 5, max difference between ranks = 4, # of suits $\neq 1$;
- *Flush*: # of suits = 1, max difference between ranks $\neq 4$.

At this level, PRL was able to identify all the correct rules achieving a BACC of 100% in all the tasks.

We have already demonstrated that the value of the game monotonically increases at any iteration of PRL. This fact is further confirmed by the plots in Figure 5. The plots show how the value of the game changes until iteration 50. In both cases the maximum value had been reached since PRL already had discovered the best rules.

In particular, Figure 5 regards the *Straight* classification and it can be noticed that there have been 3 quick changes in the value: at iteration 16 (red dashed line), 39 (green dashed line) and 46 (grey dashed line). Until iteration 16 the set of rules was immature to make any analysis. At iteration 16, the 8 best rules were the following: max(diff(ranks)) $= 5, 6, \ldots, 11$, to support the negative class. These rules represent almost all (12 is missing) the possible maximum differences for the ranks greater than 4, that is actually the value useful to identify the Straight. The only rule for the positive class was # ranks $= 5$, which is correct since in a Straight all ranks are different.

Then, at iteration 39, PRL found: max(diff(ranks)) $= 5, 6, \ldots, 12$ for the negative class and # ranks $= 5$ for the positive one. So the only difference is the inclusion of the 12 in the maximum difference between ranks. Even though the overall rule is correct, it is still not optimal because, in order to express the positive rule max(diff(ranks)) $= 4$, PRL discovered the same concept but using a bunch of negative rules.

Finally, at iteration 46 the right set of rules has been found solving the task perfectly, that is, max(diff(ranks))$= 4$ and # ranks $= 5$ for the positive class, and # suits $= 1$ for the negative one, that excludes the two cases in which the Straight is also a Flush or a Royal flush. Here, the *bias* rule has been also extracted to represent all the other cases when the two rules for the flush are not satisfied.

On the *Flush* task, the value of the game had a similar behaviour as for the *Straight*. In this case the value drastically jumped only twice: at iteration 8 (red dashed line) and 14 (green dashed line). At iteration 8 the discovered rules, despite being correct, were a bit "chaotic": max suit = 2,3 and # of suits = 2,3 for the negative class, as well as max(diff(ranks)) $= 4$ (to exclude a straight) and $\#\heartsuit, \#\diamondsuit, \#\clubsuit, \#\spadesuit = 1$. These rules are all correct to exclude a flush, in fact, they require a suit with 5 cards and hence a unique suit in the hand. For the positive class the only rule extracted was max rank = 1, which is also correct because a flush implies that all ranks in the hand are unique. Nevertheless, some iterations later (14) PRL found the optimal set of rules, that is, max(diff(ranks)) $= 4$ for the negative class, and max suit = 5 for the positive. Similarly to the *Flush* case, the *bias* rule has been extracted by PRL to include all the cases in which the hand does not contain a straight.

7.1.3. The `breast-cancer` dataset

The Wisconsin Breast Cancer dataset (`breast-cancer`) is a standard UCI [23] dataset that contains 682 hospital patients' values captured via a Fine-needle aspiration test. Each patient is described by 9 attributes concerning breast tumoral cells. The task consists in classifying a tumor between benign or malignant. The classes distribution is 35% benign, and 65% malignant.

Compared to other testbeds, `breast-cancer` can be considered a real-world dataset in which simple rules are unable to completely describe whether the tumor is malignant or not. Thus, it is not possible to compare retrieved rules w.r.t. a given ground truth. To evaluate PRL, it was therefore necessary to compare it with other rule extractions algorithms, as proposed in [24]. In particular, the quality of our approach was assessed by applying retrieved rules directly on the dataset and comparing the accuracy obtained by different approaches. Note that, differently from previous experiments, in this case, the splitting between training and test set was 90-10%. This evaluation procedure has been chosen because, for each model, we only had at our disposal the set of extracted rules after a 10-fold cross validation procedure. To train PRL, rules of degree 2 on the set of relations $\{\leq, \geq\}$ have been used. In Table 3 the achieved results are summarized, while Table 4 shows the extracted rules of each method.

Figure 6 highlights an interesting observation on PRL applied to `breast-cancer`. The figure shows performances achieved by PRL when only a subset of rules is considered. The size of the subset of rules goes from 1 up to 50. The first three rules are not enough to get good results: they are likely associated with statistical occurrences on the dataset and are unable to capture insight over the data. This changes with the fourth rule, which, alone, is able to achieve more than 92% of accuracy. The first four rules together are able to exceed 95% of accuracy, while the remaining rules are used to classify outliers and harder examples and produce an almost monotonic increase in accuracy. Overall, the PRL approach is able to achieve 99.56% of accuracy, using 50 rules. Being a model with 50 rules hardly interpretable, in Table 3 we present a comparison of a PRL with a strongly limited set of rules (up to 10), against other approaches. As highlighted in Table 3, PRL is able to achieve the best accuracy, using the 10 most

Method	Ref	# Rules	Accuracy (%)
SSV	[25]	3	86.36
GASVM	[26]	2	90.03
C-MLP2LN	[25]	5	96.92
QSVM-G	[27]	12	96.48
ReRXJ48	[28, 24]	4	94.28
PRL only 4th	-	1	92.67
PRL@5	-	5	96.12
PRL@10	-	10	**97.95**

Table 3: Accuracy of the rules extracted by the different algorithms. The highest accuracy is highlighted in **bold**.

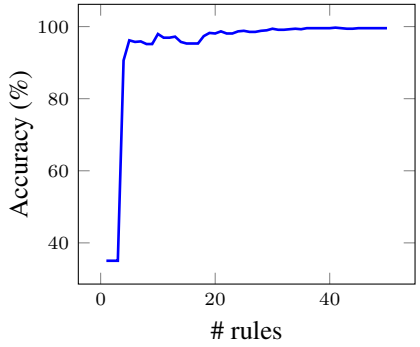


Figure 6: Plot of the accuracy on `breast-cancer` w.r.t. the number of considered rules during the classification.

relevant rules. Observe that, even with a lower number of rules (i.e., 5) the results achieved by PRL are still very good, exceeded only by C-MLP2LN and QSVM-G.

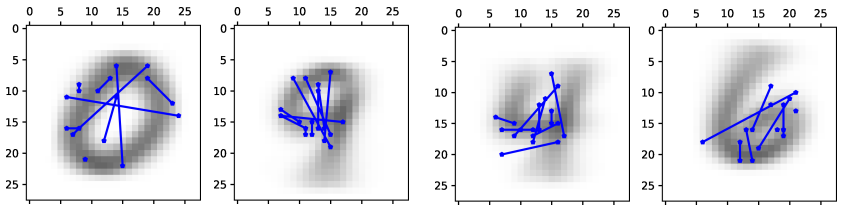
7.2. Visualization: `mnist` dataset

The `mnist` dataset is one of the most widely used dataset for the hand-written digit classification task. The digits are stored in a grey scale 28 by 28 pixel matrix, where each pixel can assume a value between 0 and 255 (0-1 normalized). The task is to recognize the digit represented by an instance. For the purpose of this experiments we perform all possible 1-vs-1 binary classifications between digits.

Akin `breast-cancer`, the classification in `mnist` cannot be done through simple rules. To present the PRL in an interpretable fashion, we aim to show how the most relevant visual features are leveraged by the model to distinguish examples belonging

Method	Rules	Class
C-MLP2N	CT$< 6 \wedge$ UCSH$< 3 \wedge$ BC< 8	M
	CT$< 9 \wedge$ MA$< 4 \wedge$ BN$< 2 \wedge$ BC< 5	M
	CT$< 10 \wedge$ UCSH$< 4 \wedge$ MA$< 4 \wedge$ BN< 3	M
	CT$< 7 \wedge$ UCSH$< 9 \wedge$ MA$< 3 \wedge 4 \leq$ BN$\leq 9 \wedge$ BC< 4	M
	$3 \leq$ CT$\leq 4 \wedge$ UCSH$< 9 \wedge$ MA$< 10 \wedge$ BN$< 6 \wedge$ BC< 8	M
SSV	MA$> 2.5 \wedge$ BC> 2.5	M
	MA$> 2.5 \wedge$ BN$> 3.5 \wedge$ BC> 0	M
	UCSI$> 5.5 \wedge$ MA$< 2.5 \wedge$ BC> 1.6	M
GASVM	CT$< 7.09 \wedge$ UCSH $< 7.91 \wedge$ SECS $< 9.76 \wedge$ BC< 6.06	B
	UCSH$< 7.7 \wedge$ BN$< 9.41 \wedge$ BC$< 6.12 \wedge$ M< 7.43	M
QSVM-G	CT$< 10 \wedge$ UCSH $< 9.95 \wedge$ BN< 6.93	B
	CT$< 7.00 \wedge$ UCSI$< 5.97 \wedge$ SECS$< 4.97 \wedge$ BN$< 4.94 \wedge$ NN< 9.94	B
	UCSH$> 2.97 \wedge$ BN> 4.94	M
	CT$> 4.96 \wedge$ UCSI> 4.00	M
	UCSI> 4.98	M
	CT$> 2.984 \wedge$ BN> 6.93	M
	CT$> 5.98 \wedge$ UCSI$> 3.00 \wedge$ UCSH> 3.99	M
	UCSH$> 2.97 \wedge$ SECS> 4.97	M
	NN> 8.96	M
	UCSI $< 3.00 \wedge$ UCSH$> 3.99 \wedge$ SECS< 4.97	B
	CT$< 2.98 \wedge$ UCSH$< 4.95 \wedge$ BN> 9.95	B
	CT$> 10.00 \wedge$ UCSI$< 3.00 \wedge$ BN< 7.96	B
ReRXJ48	BN = 1	B
	CT$\leq 4 \wedge 1 <$BN≤ 6	B
	CT$\leq 4 \wedge$ BN> 6	M
	CT$> 4 \wedge$ BN> 1	M
PRL	MA$\geq 2 \wedge$ UCSI≥ 5	M
	BN$\geq 3 \wedge$ SECS≤ 4	M
	MA$\leq 2 \wedge$ SECS≥ 3	M
	CT$\leq 6 \wedge$ MA≤ 5	B
	NN$\leq 2 \wedge$ SECS≥ 2	B
	SECS$\leq 2 \wedge$ MA≥ 3	M
	BN$\geq 6 \wedge$ BC≥ 4	M
	BN$\geq 5 \wedge$ NN≤ 1	M
	UCSH$\leq 3 \wedge$ BC≥ 4	M
	CT$\leq 6 \wedge$ NN≤ 8	B

Table 4: Rule extracted by the rule extraction algorithms reported in (Hayashi and Nakano 2015) and PRL. The class column indicates whether the rule define the positive class (M) or the negative class (B).



(a) 0 versus 9 (b) 4 versus 6

Figure 7: Visualization of the most relevant polynomial features of degree 2. The polynomial features are visualized as segments limited by the involved input variables. The left hand side plot shows the features relevant to discriminate the (a) 0 from the 9 and (b) 4 from the 6. Viceversa in the right hand side plots.

to either a class or the other.

Experiments have been performed using polynomial features of degree 2. Figure 7 illustrates two examples of the most relevant features used by the model to distinguish: (a) 0 from a 9 (left) and viceversa (right); (b) 4 from a 6 (left) and viceversa (right).

Each feature is presented using a segment between the two features (i.e. pixels) that concur to the decision process, in each rule (i.e. monomial). The background represents the average digit of the depicted class.

Plots presented in Figure 7(a) show how curvatures are used to distinguish the 0 from the 9. In particular, the algorithm looks for a "big" curvature to recognize elements belonging to the 0 class, and a smaller one to identify the 9. Figure 7(b) depicts a similar behaviour to distinguish 4 and 6. Again, the 6 is characterized by curvatures, while the horizontal dash is considered to be the important aspect to recognize the 4.

Figure 8 shows how the value of the game has changed over the iterations of PRL, while Figure 9 shows the full set of digit-vs-digit plots. In the rows there are the preferred class, while in the column the not preferred one w.r.t. the corresponding class in the rows.

7.3. Feature Selection

This set of experiments aims at assessing the effectiveness of PRL on datasets with many noisy and redundant features. The chosen testbeds have been the datasets of the NIPS 2003 Feature selection challenge [29]. All datasets are freely available at the NIPS 2003 Feature selection challenge site, `http://clopinet.com/isabelle/`

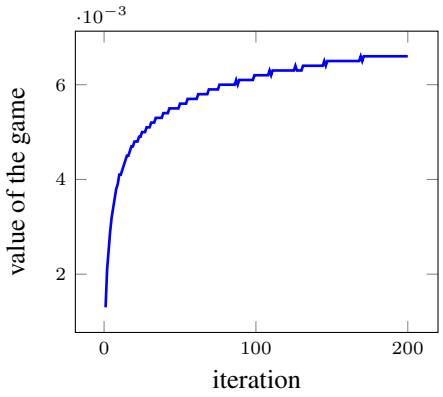


Figure 8: Value of the game w.r.t. the iteration of PRL on the `mnist` dataset.

Dataset	#Inst.	#Feats.	# Real feat.	Class prior
`dorothea`	1150	100k	50k	90/10
`gisette`	7k	5k	250	50/50
`madelon`	2.6k	500	20	50/50

Table 5: Datasets information: name, number of instance, number of features, number of relevant features (probes), and class prior.

`Projects/NIPS2003/`. Further details about the datasets are reported in [29] and [30]. A common characteristic of these datasets is the huge number of features compared to the number of training instances. All datasets consist of binary classification tasks. Table 5 summarizes the characteristics of the used datasets.

We compared PRL with standard soft-margin SVM and Random Forest Classifier. Given the huge number of features of the target datasets, the linear kernel turned out to be a good kernel for these tasks, with the exception of `madelon` in which the degree 2 homogeneous polynomial was the best performing kernel for SVM. Moreover, we evaluated the PRL using the Decision path rules generator (dubbed *PRL-RF*). The C hyperparameter of the SVM has been validated in the set of values $\{10^{-4}, \ldots, 10^{5}\}$ using a 5-fold cross validation procedure. Experiments have been performed using a 70-30% training and test split. The Random Forest Classifier has been tuned selecting the hyperparameter associated with the number of estimators in the set $\{10, 10^2, 10^3, 5000\}$, the maximum depth in $\{2, 5, 10, \text{until pure leaves}\}$ and the criterion between $\{\textit{Gini}, \textit{entropy}\}$ and validating them via 5-fold cross validation. In Table 6 the results achieved by both

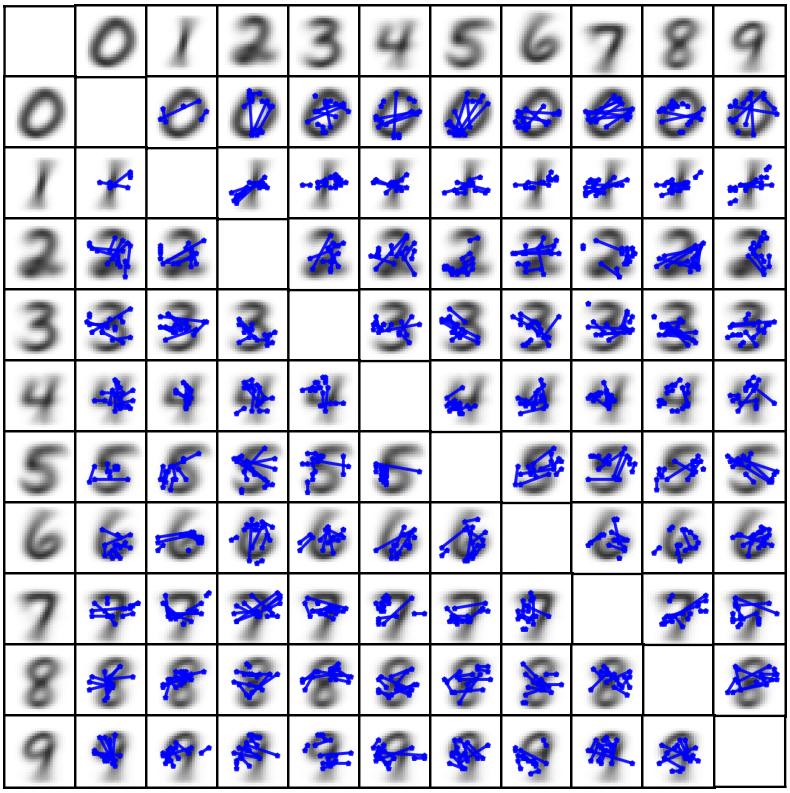


Figure 9: Depiction of the relevant features extracted by PRL for each possible pair of digits.

methods as well as the number of relevant features according to PRL are summarized.

As evident from the table, the proposed method is able to achieve better performance than SVM. It is worth to mention that, generally, the number of features used by PRL was orders of magnitude less than the number of original features. Both PRL and PRL-RF have higher (or comparable) performance w.r.t. SVM and RF. It is interesting to observe that PRL-RF consistently achieves better performance than PRL, and this can be ascribed to the quality of the features. Another observation that is worth to mention is that PRL-RF consistently has better performance than RF (the same cannot be said for PRL) and this underline the effectiveness of the feature selection capability of PRL.

Dataset	SVM	RF	PRL	PRL-RF	# Relevant/Tot. feat.
dorothea	91.88	78.13	92.69	**93.33**	500/100k
gisette	96.71	97.22	97.19	**97.67**	900/5k
madelon	60.10	71.05	62.75	**74.49**	1225/250k

Table 6: Accuracy results achieved by SVM , RF, PRL and PRL-RF. The last column indicates the number of support preference-feature pairs used by PRL. The best results are highlighted in **bold**.

8. Related work

In this section we discuss related work to the proposed method. We give particular attention to game theoretical concepts related to machine learning especially to large margin methods, and on-line (non linear) feature selection.

8.1. Game theory and machine learning

Connections between large margin methods and game theory have been already discussed in literature. In [5], Couellan investigates connections between supervised classification and generalized Nash equilibrium problems. Specifically, the geometrical properties of the separation hyperplane of SVM in the dual space are exploited to formulate a non-cooperative game. The intuition behind this game theoretical formulation is that the two players are associated with the positive and the negative class, respectively. The goal of each player is to "pull" the hyperplane close to himself. In the paper, the proposed formulation is then extended for the multi-class setting. Similar observation has also been done in [4] in which hard-margin SVM is cast into a two-player zero some game. Starting from this observation, authors propose a kernel-based method for the direct optimization of the margin distribution.

In [31], Polato et al. propose a preference learning framework inspired by game theory for multi-class classification problem. The framework defines a single optimization problem related to the optimal strategies of a two-players zero-sum game. To improve the efficiency, authors propose an approximated solution which requires the sequential optimization of many sub-games.

In [32], a game theory approach for solving multi-class classification is presented. In this work pairwise classification is seen as a decision-making problem and authors show that pairwise SVM can be cast into the proposed GT framework. They also

prove that the solution of the proposed approach is equivalent to the fuzzy pairwise SVM [33]. Game theory, specifically Shapley values, is also used as a surrogate model for interpreting machine learning models. [34] propose FAE (Formulate, Approximate, Explain), a conceptual unified framework for generating and interpreting explanations. FAE generalized methods such as [35] and [36].

Under an applicative perspective, game theoretical concepts are highly exploited in the cyber-security community because it is related to the adversarial nature of an attacker, e.g., [37, 2].

8.2. On-line feature selection

One of the first proposed approaches for performing feature selection in the feature space has been [38]. In this work, Cao et al. extended *Relief* [39], a margin based feature selection approach, in kernel space by deriving a basis set in the feature space that is used to compute the distance (useful in the computation of the *nearhit* and the *nearmiss*) in the feature space using the kernel trick.

In [40], Nguyen et al. proposed a convex energy-based framework to jointly perform feature selection and non linear SVM parameter learning. Authors empirically show that the proposed method shows significant reduction of features used while maintaining classification performance.

More recently, a similar approach has been proposed by Adeli et al. [41] for early diagnosis of Parkinson's disease. The core idea behind the proposed method is the learning of different kernels for each feature, as in the Multiple Kernel Learning framework, but for each single feature. Then the optimization problem learns how to weight these kernels and the weights represent how much discriminant the features are in the feature space.

The main difference between PRL and the just mentioned approaches is that, thanks to the feature generator component, PRL has the capability of treating problems with infinitely many features. PRL, like HOSFS [42], is also theoretically suitable for dealing with streaming of features, however, its main limitation is the efficiency that it could not be ideal in settings with high throughput.

One of the biggest challenges in feature selection is dealing with large scale data in

particular with (infinitely) many input features. This is a typical scenario in real-world applications when data instances have high dimensionality or it is expensive/inconvenient to acquire all attributes. In these contexts, batch approaches are simply not applicable for computational reasons. Thus, there is the need to move towards on-line feature selection (OFS) approaches [43, 44], which can work with a small and limited number of features. For a comprehensive comparison of linear and non-linear feature selection methods we refer the reader to [45].

9. Conclusions and future work

This paper proposed a new preference learning approach for classification (and label ranking) based on game theoretical concepts. The learning problem is seen as a two-players zero-sum game solved by a novel incremental algorithm. We provided theoretical guarantees about the convergence of the algorithm as well as an extensive set of experiments demonstrating its effectiveness. We also showed the capability of PRL in identifying explanation rules for interpreting the predictions.

In the future we aim at applying PRL for extracting feature correlations. For example by creating artificial tasks where a target feature is used as label and the extracted rules describe how other features correlate with the target one. Moreover, we aim at introduce new feature generation schemes. A possibility could be to explore random feature generation methods such as the Rahimi and Recht random features [46]. Finally, we also intend to relax the PRL formulation in order to get a soft margin version of the algorithm.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems 27, 2014, pp. 2672–2680.

[2] D. P. Yufei Liu, A novel kernel SVM algorithm with game theory for network intrusion detection, KSII Transactions on Internet and Information Systems 11 (8).

[3] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1) (1997) 119–139.

[4] F. Aiolli, G. Da San Martino, A. Sperduti, A kernel method for the optimization of the margin distribution, in: Artificial Neural Networks - ICANN 2008, 2008, pp. 305–314.

[5] N. Couellan, A Note On Supervised Classification and Nash-Equilibrium Problems, RAIRO - Operations Researchdoi:10.1051/ro/2016024.
URL https://hal-univ-tlse2.archives-ouvertes.fr/hal-01354857

[6] J. Fürnkranz, E. Hüllermeier, Preference Learning, 1st Edition, Springer, 2010.

[7] S. Har-Peled, D. Roth, D. Zimak, Constraint classification for multiclass classification and ranking, in: Proceedings of the 15th International Conference on Neural Information Processing Systems, NIPS'02, MIT Press, Cambridge, MA, USA, 2002, pp. 809–816.
URL http://dl.acm.org/citation.cfm?id=2968618.2968719

[8] G. Brown, Iterative solution of games by fictitious play, Activity Analysis of Production and Allocation (1951) 374–376.

[9] M. Polato, F. Aiolli, Interpretable preference learning: A game theoretic framework for large margin on-line feature and rule learning, in: AAAI, 2019, pp. 4723–4730.

[10] I. Tsochantaridis, T. Hofmann, T. Joachims, Y. Altun, Support vector machine learning for interdependent and structured output spaces, in: Proceedings of the Twenty-first International Conference on Machine Learning, ICML '04, ACM, New York, NY, USA, 2004, pp. 104–. doi:10.1145/1015330.1015341.

[11] F. Aiolli, A. Sperduti, A preference optimization based unifying framework for supervised learning problems, in: Preference Learning, 2010.

[12] R. E. Schapire, Y. Freund, P. Barlett, W. S. Lee, Boosting the margin: A new explanation for the effectiveness of voting methods, in: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, 1997, pp. 322–330.

[13] N. J. Nilsson, Learning machines: foundations of trainable pattern-classifying systems, McGraw-Hill, New York, NY, USA, 1965.

[14] R. O. Duda, P. E. Hart, Pattern Classification and Scene Analysis, John Willey & Sons, 1973.

[15] S. Har-Peled, D. Roth, D. Zimak, Constraint classification: A new approach to multiclass classification, in: N. Cesa-Bianchi, M. Numao, R. Reischuk (Eds.), Algorithmic Learning Theory, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 365–379.

[16] J. von Neumann, Zur theorie der gesellschaftsspiele 100 (1928) 295–320.

[17] Y. Freund, R. E. Schapire, Adaptive game playing using multiplicative weights, Games and Economic Behavior 29 (1-2) (1999) 79–103.

[18] Y. Freund, R. E. Schapire, Game theory, on-line prediction and boosting, in: COLT, 1996, pp. 325–332.

[19] S. D. Bopardikar, C. Langbort, Incremental approximate saddle-point computation in zero-sum matrix games, in: 53rd IEEE Conference on Decision and Control, 2014, pp. 1936–1941.

[20] G. W. Brown, Iterative solutions of games by fictitious play, In: Activity Analysis of Production and Allocation (1951) 374–376.

[21] G. S. Kimeldorf, G. Wahba, Some results on tchebycheffian spline functions, Journal of Mathematical Analysis and Applications 33 (1) (1971) 82–95.

[22] T. Hofmann, B. Scholkopf, A. J. Smola, Kernel methods in machine learning, The Annals of Statistics 36 (3) (2008) 1171–1220.

[23] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml

[24] Y. Hayashi, S. 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 (2015) 9 – 16.

[25] D. Nauck, R. Kruse, Obtaining interpretable fuzzy classification rules from medical data, Artificial Intelligence in Medicine 16 (2) (1999) 149 – 169. doi:https://doi.org/10.1016/S0933-3657(98)00070-0.

[26] Y.-C. Chen, C.-T. Su, T. Yang, Rule extraction from support vector machines by genetic algorithms, Neural Computing and Applications 23 (3) (2013) 729–739.

[27] G. Bologna, Y. Hayashi, Qsvm: A support vector machine for rule extraction, in: Advances in Computational Intelligence, Springer International Publishing, Cham, 2015, pp. 276–289.

[28] Y. Hayashi, Y. Tanaka, T. Takagi, T. Saito, H. Iiduka, H. Kikuchi, G. Bologna, S. Mitra, Recursive-rule extraction algorithm with j48graft and applications to generating credit scores, J. Artif. Intell. Soft Comput. Res. 6 (2016) 35.

[29] I. Guyon, S. Gunn, A. Ben-Hur, G. Dror, Result analysis of the nips 2003 feature selection challenge, in: L. K. Saul, Y. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems 17, MIT Press, 2005, pp. 545–552.

[30] N. Johnson, A study of the nips feature selection challenge (2009).
URL https://web.stanford.edu/~hastie/ElemStatLearn/comp.pdf

[31] M. Polato, G. Faggioli, I. Lauriola, F. Aiolli, Playing the large margin preference game, in: Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning, Springer International Publishing, 2019, pp. 792–804.

[32] M. Petrovskiy, A game theory approach to pairwise classification with support vector machines, in: 2004 International Conference on Machine Learning and Applications, 2004. Proceedings., 2004, pp. 115–122.

[33] S. Abe, T. Inoue, Fuzzy support vector machines for multiclass problems, in: 10th European Symposium on Artificial Neural Networks, ESANN 2002, pp. 113–118.

[34] L. Merrick, A. Taly, The explanation game: Explaining machine learning models using shapley values, in: A. Holzinger, P. Kieseberg, A. M. Tjoa, E. Weippl (Eds.), Machine Learning and Knowledge Extraction, Springer International Publishing, Cham, 2020, pp. 17–38.

[35] A. Datta, S. Sen, Y. Zick, Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems, in: 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 598–617. doi:10.1109/SP.2016.42.

[36] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 4768–4777.

[37] M. Zolotukhin, T. Hamalainen, Support vector machine integrated with game-theoretic approach and genetic algorithm for the detection and classification of malware, in: 2013 IEEE Globecom Workshops (GC Wkshps), 2013, pp. 211–216.

[38] B. Cao, D. Shen, J.-T. Sun, W. Yang, Z. Chen, Feature selection in a kernel space, in: International Conference on Machine Learning, IMCL'07, 2007.

[39] I. Kononenko, Estimating attributes: Analysis and extensions of relief, in: F. Bergadano, L. De Raedt (Eds.), Machine Learning: ECML-94, Springer Berlin Heidelberg, Berlin, Heidelberg, 1994, pp. 171–182.

[40] M. H. Nguyen, F. de la Torre, Optimal feature selection for support vector machines, Pattern Recognition 43 (3) (2010) 584 – 591. doi:https://doi.org/10.1016/j.patcog.2009.09.003.

[41] E. Adeli, G. Wu, B. Saghafi, L. H. An, F. Shi, D. Shen, Kernel-based joint feature selection and max-margin classification for early diagnosis of parkinson's disease, in: Scientific reports, 2017.

[42] S. Sandhiya, U. Palani, A novel hosfs algorithm for online streaming feature selection, in: 2020 International Conference on System, Computation, Automation and Networking (ICSCAN), 2020, pp. 1–6. doi:10.1109/ICSCAN49426.2020.9262401.

[43] J. Wang, P. Zhao, S. C. H. Hoi, R. Jin, Online feature selection and its applications, IEEE Transactions on Knowledge and Data Engineering 26 (3) (2014) 698–710.

[44] X. Wu, K. Yu, W. Ding, H. Wang, X. Zhu, Online feature selection with streaming features, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (5) (2013) 1178–1192.

[45] O. Krakovska, G. Christie, A. Sixsmith, M. Ester, S. Moreno, Performance comparison of linear and non-linear feature selection methods for the analysis of large survey datasets, PloS one 14 (3).

[46] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Proceedings of the 20th International Conference on Neural Information Processing Systems, NIPS'07, 2007, pp. 1177–1184.