Computers and Mathematics with Applications 66 (2013) 1892–1904

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Using reinforcement learning to find an optimal set of features

Seyed Mehdi Hazrati Fard *, Ali Hamzeh, Sattar Hashemi

Department of Electrical and Computer Engineering, Shiraz University, Shiraz, Iran

ARTICLE INFO

Keywords:
Feature selection
Markov decision process
Reinforcement learning
Temporal difference

ABSTRACT

Identifying the most characterizing features of observed data is critical for minimizing the classification error. Feature selection is the process of identifying a small subset of highly predictive features out of a large set of candidate features. In the literature, many feature selection methods approach the task as a search problem, where each state in the search space is a possible feature subset. In this study, we consider feature selection problem as a reinforcement learning problem in general and use a well-known method, temporal difference, to traverse the state space and select the best subset of features. Specifically, first, we consider the state space as a Markov decision process, and then we introduce an optimal graph search to overcome the complexity of the problem of concern. Since this approach needs a state evaluation paradigm as an aid to traverse the promising regions in the state space, the presence of a low-cost evaluation function is necessary. This method initially explores the lattice of feature sets, and then exploits the obtained experiments. Finally, two methods, based on filters and wrappers, are proposed for the ultimate selection of features. Our empirical evaluation shows that this strategy performs well in comparison with other commonly used feature selection strategies, while maintaining compatibility with all datasets in hand.

1. Introduction

Feature selection (FS), a beneficial preprocessing step, is usually performed in order to reduce the dimension of data. In contrast to feature extraction methods, in feature selection approaches, the meanings of the features remain intact while the feature space is optimally reduced according to a certain assessment criterion [1]. There are many potential advantages of using feature selection, including reducing the storage requirements, reducing the time complexity of the problem in hand, overcoming the curse of dimensionality to improve prediction performance, facilitating data visualization to understand the underlying structure of the data [2], and above all removing the irrelevant features according to a certain criterion.

In the past, we rarely faced problems in which the number of features exceeds 40 [2]. However, in recent years, with the advance of science and technology, datasets have grown hugely and now include large numbers of features. As examples, one can refer to a dataset related to selecting genes from microarray data or another dataset related to text categorization; both of them have a large number of features [2]. In several cases, the problems might contain instances with a large number of similar features representing the same properties, while indeed most of them are not worth considering. To be specific, they may have overlaps and hence reveal the same information in the field of concern, i.e. they are redundant. Therefore, the importance of selecting features, which leads to lowering the complexity of the classifier, becomes greater as the number of features increases.

* Correspondence to: Ghodusi Gharbi st., Mabas apartments, block 802, unit 23, Shiraz, Iran. Tel.: +98 711 6133160.
E-mail addresses: hazrati@cse.shirazu.ac.ir, smhf525@yahoo.com (S.M. Hazrati Fard), ali@cse.shirazu.ac.ir (A. Hamzeh), s_hashemi@shirazu.ac.ir (S. Hashemi).

0898-1221/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.camwa.2013.06.031

In general, FS methods are divided into three main categories: filters, wrappers, and embedded approaches [2]. Filter approaches usually work independently of the chosen predictor, while the wrappers utilize the learning machine of interest as a black box to score subsets of features according to their predictive power [2]. Meanwhile, embedded methods such as decision trees perform feature selection in the training phase.

In this work, a reinforcement learning (RL) based method [3] is developed to select the best features and reduce the generalization error. Specifically, we consider the problem of FS as an RL problem, so it is not dependent on a specific dataset, because the RL methods use a pragmatic way to search the environment and can be adapted to any conditions.

Taking inspiration from the upper confidence tree (UCT) [4], a graph-based method, namely the upper confidence graph (UCG) [5], is introduced to restrict the exploration and exploit all previous experiences. In this way, using the experiences that have been gathered so far, a temporal difference (TD) based framework is embodied to traverse the state space and select the best next feature. Finally, an algorithm named *FSTD* (Feature Selection using Temporal Difference) is introduced. An evaluation function is required to estimate the merit of each state, according to the selected features. It is better that this evaluation function be the same as that used in the final assessment of the work, because it is probable that a subset of features is defined proper for discrimination by a classifier, but another classifier does not prefer this subset. In this way, the SVM [6] as a robust classifier for binary classes is considered here.

In order to use an RL framework, any subset of features is considered as a state and an action is such as taking a new feature in this state, and passing it to a new state. To indicate the influence of taking a feature as an action, the difference between rewards of two consecutive states is employed, and the average of collected rewards for that feature in several iterations is considered as its final score. In the case that a dataset has too many samples, it is favorable to run an instance reduction approach at the beginning and pick up only the selected instances for evaluation.

A critical problem in state of the art feature selection approaches is recognizing the proper number of selected features. In this work, an applicable solution is proposed to detect the proper number of selected features. Two methods, namely *F-FSTD* and *W-FSTD*, inspired from filter and wrapper approaches are introduced to select the final subset of features. Experimental results demonstrate acceptable accuracy of our method, while having a reasonable complexity. Our main contributions in this paper are fourfold.

- We extend the work done in [7] by introducing RL-based frameworks to conduct an optimal search in the state space. Since the state space of the problem is so huge, the dynamic of the environment is not reachable and we use some episodes to reconnoiter the problem area.
- We introduce an *AOR* (average of rewards) criterion to assess how good a feature is according to different conditions. The difference between the values of two consecutive states indicates the effectiveness of the corresponding feature that causes this transition. The average of these values in several iterations is defined as the *AOR* of this feature.
- In addition to addressing the problem as a one-player game, an iterative algorithm is employed to traverse the state space and finally select the best subset of features in two different ways: a filter-based approach and a wrapper-based approach. The experimental results demonstrate the superiority of these methods compared with the rival methods.
- Selecting features with a wrapper-based approach has its merit in pragmatic problems when the construction of samples is costly. In this situation, a suboptimal subset of features with a low complexity is sufficient.

The remainder of this paper is organized as follows. Section 2 defines the FS problem. In Section 3, related works are reviewed. The proposed method is explained in detail in Section 4. The experimental results are discussed in Section 5, and finally, in Section 6, we conclude the paper and present some perspectives for future works.

2. Problem statement

Feature selection is a useful preprocessing step for most data mining methods. The objective of feature selection is generally to provide faster and more cost-effective predictors, improving the performance of the predictors, and providing a better understanding of the underlying distribution of the data [2]. This paper focuses on feature selection in the realm of supervised learning. In this field, the induction algorithms are typically presented with a set of training instances, where each instance is described by a vector of features and a class label. The ultimate goal is to select some preferable features while ignoring the rest [8].

As an empirical example, consider a medical dataset representing some features for some persons. The features might include the age, weight, blood pressure, and some other attributes of a patient; as we do not know where these data will be used in the future, we have to save all of them. Suppose that the class label might indicate whether or not the person has a specific illness. To do this, it is necessary to realize which features are relevant to this subject and then detect if some of them are irrelavant or redundant that they have the same information. The task of the induction algorithm is to define a classifier that will be useful in classifying future cases. The classifier is a mapping from the space of feature values to the set of class labels [8].

The problem of FS is that of selecting a proper subset of the original features of a certain dataset, such that an end learner executing on data containing only these features generates a classifier with the highest possible accuracy; so FS chooses a set of features from all of them, and does not construct new ones [8]. The various FS methods, classically categorized into filters, wrappers, and embedded approaches, each have their pros and cons, regarding both conditions of the dataset and also the aim and scope of the problem.

3. Related works

There exists extensive research on the realm of FS, classically categorized into three main groups in the literature: filters, wrappers, and embedded methods. In this section, we describe these categories in detail and introduce some of the well-known approaches related to them.

Filtering methods: Filters usually select subsets of variables as a preprocessing step, independently of the chosen predictor [2]. Filtering features means either ranking or selecting the best subset of features; thus it is necessary to define a search strategy to select the best subset of features [1]. In this way, features are ranked regard to the relevancy between them and class labels. Methods of feature ranking consider each feature in an independent way, trying to determine how useful it may be. Unfortunately, filters do not take into account the feature redundancy and they neglect the importance of the overlapping concept. Filters usually score the features based on a distance between classes, i.e., dissimilarity measures, or based on correlation and information-theoretic measures, i.e. similarity measures [6].

Some typical filter methods include information theory [9], data variance [2], Pearson correlation coefficients [10], Fisher score [11], and the Kolmogorov–Smirnov test [12]. Also, a well-known method in this area is RELIEF [13]. which measures the feature impact on each instance vicinity. This neighborhood, defined from the Euclidean distance built from all features, might however be biased due to feature redundancy.

Generally, feature ranking is not optimal; nevertheless, it may be preferable to other FS methods because of its efficient computation [2].

Wrapper methods: Most of the feature selection methods are based on wrappers that consider an end learner as a black box to score subsets of features based on their predictive strength [2]. The wrapper-based techniques evaluate the subset of features using the learning hypotheses that will ultimately be employed. In its most general formulation, the wrapper methodology depends on using the prediction performance of a given learner to evaluate the relative merits of feature subsets [2]. Therefore, they "wrap" the selection process around the learning algorithm. In this way, a proper method to search in the environment is an important issue, because it is very important how to navigate in the feature powerset lattice, and deal with the Exploration versus Exploitation dilemma (EvE), meaning that we are not able to determine exactly when the time is over for exploration prior to exploitation. Greedy search strategies seem to be computationally advantageous and robust against overfitting.

To overcome the EvE dilemma, hill-climbing approaches lead to a useful method named correlation-based feature selection (CFS) [14]. Combining global and local search operators [15], mixing forward and backward selection [16], using look-ahead or making large steps in the lattice [17], among many other heuristics, have also been proposed to address the EvE dilemma [7].

Wrappers are considered as the best, among the other methods; however they suffer from the main problem, i.e. their high complexity.

Embedded methods: These methods perform variable selection in the training phase and are usually specific to a given learning machine [2]; in other words, they exploit the learned hypothesis and incorporate feature selection as part of the training process [18]. They converge faster to a solution by avoiding retraining a predictor from scratch for every feature subset investigated, and they make better use of the available data without the need to split the training data into a training set and a validation set [2].

Decision trees such as CART have a built-in mechanism to accomplish FS. A famous approach in this field uses random forests [19] to compute feature scores. The main problem with these scores is that they are hindered by feature redundancy. The embedded approaches also suffer from the EvE dilemma.

To sum up, filtering and embedded approaches consider each feature in a fixed context. Moreover, they do not capture feature correlations, which may lead to the feature redundancy. Wrapper approaches, on the other hand, select the best feature subset with high computational complexity, which makes them not tractable in the presence of many features.

Beyond the simple basis of the most FS methods, the problem can be seen in a special way, and some innovative methods can be introduced to select the best subset of features. Here, a new approach is introduced which uses a pragmatic manner to select the best features and is adaptive to all conditions. The point is that it learns directly from the experience and is not dependent on a specific dataset. In this way, two methods based on RL [3] are used for selecting the best features. To do this, the state space must be introduced as a Markov decision process (MDP). Then any subset of features is considered as a state, and selecting an unobserved feature in that state is an admissible action in this situation; so the problem is introduced as a one-player game, and the benefits of RL methods can be used [7].

The FS problem is formalized as follows. Let ψ stands for the set of all features and let F be a subset of ψ, i.e. a state in the state space. Indeed, the state space is the power set of ψ. An action could be the selection of an unseen feature (f from ψ) and is regarded as a transition from one state to another. Each iteration starts from an empty subset, adds a selected feature to the previous set [7], and proceeds until the stopping condition is satisfied (we will investigate this condition later, in Section 4.5.)

In RL approaches, the policy π is the procedure of exploration in the environment and selecting the best action in the current state to reach the best regions. Mapping these assumptions to the FS problem leads to selection of the best feature in each state and transferring it to a new state. Based on RL methods [3], the ultimate objective is to exploit the experiences

and determine the best policy π^* that leads to selecting the best reward according to the following formula:

$$\pi^*(F) = Arg \max_{f \in \psi/F} (rewards), \tag{1}$$

where $\pi^*(F)$ is the best policy of selecting a feature f from the set of all features ψ except those contained in the state F. The motivation behind the presented MDP framework is to select a policy with optimal behavior. In order to find the maximum reward and transition probability functions that lead to traversing another state, the most promising regions in the state space are investigated. Given that the goal is to minimize the generalization error of the learned hypothesis from the training set, the best policy can be shown to be

$$\pi^* = Arg \min_{\pi} \ Error(A(F_\pi)), \tag{2}$$

where A is the learned hypothesis according to the problem [20].

4. The proposed framework

In the initial works done in the field of FS using the RL method [7], a Monte Carlo (MC) approach was employed, taking inspiration from [20]; a tractable approximation of the optimal policy was provided. The proposed policy approximation relied on the Upper Confidence Tree (UCT) framework [4], which controls the optimal exploration versus exploitation tradeoff of the Upper Confidence Bound (*UCB*) algorithm [21]. While an MC tree is employed, some episodes were used to investigate the environment. At the end of each episode, an evaluation of the path is performed, and the reward obtained is passed to all states of this traverse.

In this work, another method based on RL is introduced: the temporal difference (TD) method [3] is one that can traverse the state space and experience the environment to solve the prediction problem. This method makes use of the benefits of dynamic programming (DP), but it does not rely on the underlying dynamics of the environment. Furthermore, it has the advantages of MC approaches, while preventing the accumulation of calculation. In MC approaches, an episode is considered and the process is postponed to the end of path. Then the return value is calculated as an evaluation of the final state, and it is passed to all states of path to update their values. In the TD method, to update the value of the current state it is necessary to select the best action of this state and move to the next state instead of waiting until the final state is reached. The simplest TD method is known as TD(0), and the target for this method is to update the value of the current state as follows [3]:

$$V(S_t) \leftarrow V(S_t) + \alpha \ [r_{t+1} + \gamma V(S_{t+1}) - V(S_t)], \tag{3}$$

where $V(S_t)$ and $V(S_{t+1})$ are the values of the current and next state respectively, r_{t+1} is the reward that is gained in this transition, α and γ are constants between 0 and 1, α is used to control the rate of update, and γ is a discount factor to moderate the effect of observing the next state. When γ is close to 0, there is a shortsighted condition; when it tends to 1 it exhibits farsighted behavior [3].

The *UCB* method [21] proposed in the literature explores the state space and moderates the effect of exploration in the case that exploitation is necessary. An extended version of this work, namely UCG [5], has been also introduced to improve the applicability of search in the state space where the search space was imagined as a graph to better utilize the experiences. For the sake of self-containedness of this paper, these approaches are explained here.

4.1. Upper confidence graph (UCG)

Here, a state in the *UCB* tree is regarded as a node, and it could be considered in two phases.

The bandit-based phase: This phase is used when the algorithm has reached a node that has been observed before, so the node preserves a set of performed actions (hereinafter called repository) in transmitting it to another state. In this stage, one of these actions that maximizes the *UCB* criterion could be selected with a high probability:

$$a^* = Arg \max_{a \in A} \left\{ \hat{\mu}_{F,a} + \sqrt{c_e \ln(T_F)/t_{F,a}} \right\}, \tag{4}$$

where T_F is the number of times that node F has been visited, $t_{F,a}$ is the number of times action a has been selected in node F, $\hat{\mu}_{F,a}$ is the average of the collected reward when selecting action a in node F, and finally c_e is a parameter of the algorithm that is a small positive number in the original formula [21] and experimentally has been set to 2, with the aim of controlling the exploration strength [4]; it takes different values according to each dataset in some papers [7].

This formula is divided to two parts and according to their importance uses an adaptive coefficient to combine them [5]. In the first part, we consider the average reward that has been gathered when the action a has been selected in a state, and the second part is the residual portion of the *UCB* formula. Instead of using several values according to various conditions, it could be inferred from experience that the first part is more important than the second part. Therefore, the parts of the formula could be normalized to the ranges [0, 10] and [0, 1], respectively, and be agglomerated as a simple addition. This

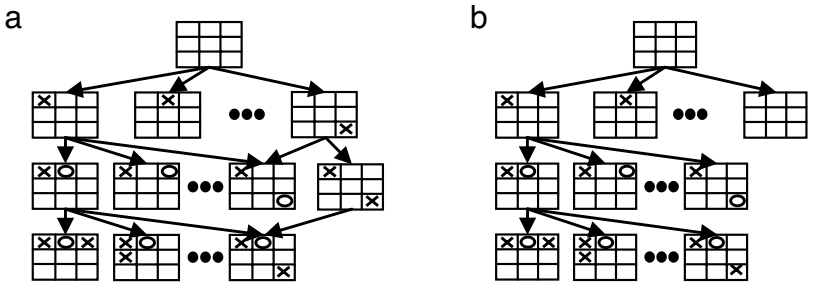

Fig. 1. The state space of a simple example, a Tic–Tac–Toe game. (a) Tree search. (b) Graph search.

criterion is called *Adaptive-UCB*. Thus, there is not any need to use several coefficients for various datasets, as in

$$a^* = Arg \max_{a \in A} \left\{ Norm1[\hat{\mu}_{F,a}] + Norm2 \left[\sqrt{c_e \ln(T_F)/t_{F,a}} \right] \right\}. \tag{5}$$

According to Eq. (5), each action from the repository that has collected more average of rewards when transferring from this node to another has more chance to be selected in this situation. Moreover, if this action has been selected more than others in this state, it will have less chance to be selected again. As mentioned, this procedure is performed with a high probability, but, to prevent stopping the exploration, an opportunity is considered to select a new feature from the features that neither has been seen in this state nor is an action of this state. Then a new action is selected in this state with a low probability from the actions that have the best operation so far [7]. This is like the ε-greedy method in RL [3], and the exploration will not be stopped in any conditions, so the next feature could be selected based on a heuristic such as the collected rewards corresponding to this feature in the previous iterations.

Selecting features in this way continues until a new node is reached and then it must switch to the random-based phase. If the stop criterion is satisfied, this iteration has been terminated and the algorithm passes to the next iteration.

The random-based phase: Whenever an unseen state is met, it is added to the tree as a new node, and, because of the lack of experience about this node, use of the bandit-based phase is stopped. Now the next feature could be selected randomly, and it is time to explore freely in the state space and traverse several regions. In the original algorithm, this phase continued until reaching a final node and one could not return to the exploit-based mode [7]. In its improved version, UCG, it can switch back to the bandit-based phase. It will continue selecting features in this way and may switch between these two phases several times, until the stop condition is satisfied [5].

The procedure of switching between these two phases is depicted in a simple example in Fig. 1. Consider the Tic–Tac–Toe game having a 3 by 3 environment and two players, with their marks X and O. One player starts the game and put a mark in an empty place, and then it is the turn of the other player. The game goes on until three similar marks are aligned in a row, column, or diagonal, or all places are occupied. The state space of the game could be as Fig. 1(a). As can be seen, exploring in these states is too complex, especially when a state is visited through two different paths; in fact, there is no access to the previous experiences. A solution to this problem is to formalize the state space as a graph and eliminate the repetitive nodes, as in Fig. 1(b). Then it can switch between bandit-based and random-based phases according to the conditions. Due to this fact, not only is the search space minimized, but also the explored space and the previous experiences are now reachable [5].

In this work, based on the above phases, two operations, the *Exploit-based* function and the *Explore-based* function, are introduced to explore the state space and exploit the experiences that have been gathered so far. To determine which of them is appropriate in each situation, a test is performed to compare the current state with the nodes of search graph and to check if this node has been met before. Regardless of whether there are any experiences about this node, we will use *Exploit-based* or *Explore-based* functions, respectively; they are discussed in Section 4.6, but first we need to explain some concepts.

4.2. The reward function

A primary requirement in the temporal difference method is to set up a subtle reward function that considers both efficiency and tractability [3]. Hence, to estimate the generalization error, the SVM classifier [6] will be the choice, but, since this function is used a lot, it should not be too computationally complex. Since the complexity of SVM depends on the number of samples, instance reduction is very useful in this case. Whereas the TD method needs evaluation in each state to specify its path, instance reduction could be considered as a preprocessing step, and any evaluation can be performed on the selected features.

Another method that is very convenient with low complexity is AUC [6], that is, the area under the curve of the ROC (receiver operating characteristic) [22]. First, consider the selected instances; each one is represented by $z = (x, y)$, where x is the dimension of the sample and y is the related class label. For binary classes, any selected sample in the preprocess can be labeled as negative or positive. For each positive sample, its k nearest neighbors are considered and a new criterion

is constructed [23]. The numerator initially is set to zero. Now, consider each positive sample and the k nearest neighbors around it. For a positive sample, if the number of positive samples between k nearest neighbors around it ($S_F(z)$) is more than the number of negative ones, the numerator is incremented by 1, and if the number of positive and negative samples are equal, it is incremented by 0.5. If the value of k is odd, the 0.5 score can be neglected. For example, in our implementation with $k = 5$, there never be the condition of equal numbers of negative and positive samples around an instance. The denominator is the product of the number of positive samples of V and the number of negative samples [7]. Eq. (6) illustrates this criterion.

$$V(F) = \left|\left\{(z, z') \in V^2, y < y', S_F(x) < S_F(x')\right\}\right| / \left\{(z, z') \in V^2, y < y'\right\}. \tag{6}$$

We can use any other end learner in this way; however, it is preferred to use the same hypotheses for the reward function and the final evaluation of features, so the progress of the algorithm and the final evaluation of subsets are commensurate. Here, a Gaussian SVM [24] is used for evaluation of a state according to the present features in that state, and finally this method is employed in the final evaluation.

4.3. Preprocessing in the realm of FS

In accordance with the circumstances of a dataset, the FS procedure might need some preprocessing. For example, a preprocessing step is required when the dataset is not commensurate [2]. Then the ranges of variation in features are diverse, and, to calculate the distance of two instances, the feature with wide range would mask the others. In this condition, it is necessary to normalize the features and consider them at the fair condition that all of the features are in the analogous range and no one mask the others. A way to normalize the dataset is to scale all features in the range between 0 and 1 [6].

Another preprocess that reduces the complexity of methods is instance reduction, where some samples are selected as representative of the dataset distribution. Therefore in evaluating each state we do not need to use all samples, and the chosen examples are sufficient. One can point to the code book of the dataset, but this seems insufficient. Here, first, the dataset is clustered with a simple method like K-means [1], with an optimal value of K according to the size of dataset. Then, inspired by the methods that are proposed in the realm of reducing instances in KNN [1], and selecting some instances as indicators [25,26], we select the proper ones. Wilson states that the best instances that reveal the distribution of a cluster are the ones in the center of cluster [26], and Ritter states that the best ones are in the boundary [25]. Therefore we pick up some samples from the center of clusters and some in the boundaries. These selected samples can be used in all evaluations of TD episodes instead of using all the samples.

4.4. Feature scoring

To determine the best next action in a state, a measure is necessary to compare features. To do so, inspired from [27], we introduce a criterion named *AOR* (average of rewards). As the name implies, *AOR* is the average of rewards of each feature whenever it is selected in a state, and it is determined as

$$\mathrm{AOR}_f = \mathrm{Average}\{V(F_t) - V(F_{t+1})\}, \tag{7}$$

where $V(F_t)$ and $V(F_{t+1})$ are the assessment values of the current and successor states and f is the feature that causes a transfer from state F_t to the next one, F_{t+1}. In this criterion, the difference between the values of the current and the next state reveals the influence of the selected feature and the reward function is a Gaussian SVM [24] in accordance with the features of the states, as mentioned already.

In calculating this criterion, there is no need to maintain all rewards of a feature, whenever it is selected, since it can be computed in an incremental manner [3]. To do so, a 2 by n array can be constructed at the beginning of the algorithm; where n is the number of features. Then each column belongs to a feature; the first row indicates the number of times that this feature is selected, and the second row shows the *AOR* value of the relevant feature. All of the array cells are initially set to zero, and each time a feature is selected the visiting number is incremented by 1. The *AOR* value can be updated incrementally as

$$(\mathrm{AOR}_f)_{New} = [(k-1)(\mathrm{AOR}_f)_{Old} + V(F)]/k, \tag{8}$$

where $(\mathrm{AOR}_f)_{New}$ and $(\mathrm{AOR}_f)_{Old}$ are the current and previous values of *AOR* for feature f and k is the number of times this feature has been visited.

4.5. The stop condition

As already mentioned, each iteration needs a stop criterion to terminate. In the discussed MC method, reaching this condition is random or in accordance with the length of the episode. Here, taking inspiration from sequential search strategies, especially sequential forward selection (SFS) [1], we define a stop criterion. In the SFS algorithm, if adding a feature to the previous state worsens the value of the new state, it is time to stop adding features. As the condition of adding feature to a subset is like that in SFS, the stop condition could be similar, but since we do not examine all subsets of features in the state space, stopping can be postponed a little; then if in m consecutive times the value of the state declines, we can stop the search in this episode. Taking into account the number of features, the proper value for m can be determined. In fact, the greater this number, the more the number of states in an episode is.

4.6. The proposed algorithm

The proposed algorithm, namely *FSTD* (feature selection using the temporal difference method), covers all requirements to implement the method. The pseudo code, shown in Algorithm 1, contains four major functions, which are discussed in detail.

(1) *The main algorithm of FSTD*: Since the method is iterative, initially the number of iterations is given as input; the algorithm sets the *AOR* values and visitation numbers to zero for all features and initializes the graph in the zero level. At the beginning of each iteration, it starts from an empty state and calls a function to select a feature.

If the current state has been seen in the graph and some features have been selected here, then there are the experiences of previous visits, and the *Exploit-based* function must be called. Otherwise, the *Explore-based* function is beneficial to select a new feature. Now, the action as a selected feature is added to the current state and it transfers it to another one.

Based on the TD algorithm, each node in the search graph needs an instant value. Initially, at the creation of a node, its value is set to the fitness of the features of this state, *but* every time this node is being seen, this value is updated according to TD(0) (see Eq. (3)).

To evaluate a subset of features, a Gaussian SVM [24] is run on the selected instances with the features of this state. Every time a feature is selected as an action, the *AOR* value according to this feature must be updated. This procedure proceeds until a stop condition is satisfied. This stop condition is reached whenever the reward of a state declines m times consecutively. Eventually, after all iterations, the best features could be determined according to the final graph or the *AOR* of each feature. Algorithm 1 is the pseudo code of main *FSTD*.

Algorithm 1: The *FSTD* main body

Input: number of iterations T
Output: selected features, final graph
$\forall f \in \psi : AOR(f) = 0$
Initialize graph: G
// The current state $= F$
For iteration $= 1$ to T do
  $F = \Phi$
  While stop condition is not satisfied
    If $F \in G$
      $f = Exploit\text{-}based(F)$;
    Else $f = Explore\text{-}based(F)$;
    End If
    $F \leftarrow FU\{f\}$
  End While
End For

(2) *The Exploit-based function*: Whenever the algorithm is in the state that has been visited before in the graph, presumably some features have been selected at this node already. Taking inspiration from [28], to use fair exploration and exploitation in the state space a method has been proposed in [7]. Based on that work, we state that, if the integer part of T_F^b is incremented, the new feature f with the largest *AOR* (that has not been seen as the features or the actions of this state later) is selected, indicating that there is still a chance to explore; otherwise, *UCB* is used (Eq. (4)) to select a feature from the repository of this node and exploit the experiences. T_F is the number of times node F has been visited previously and b is a parameter of algorithm that must be a positive number less than 1 (e.g. set to 0.7 in [7]). Finally, the selected feature is passed to the main function to continue its path until the stopping criterion is reached.

At the first visitations of a state, the probability of selecting a feature from the repository of this state is low, and usually an unseen feature is selected according to its *AOR* scores, but, with the progress of iterations in the latest visitations of a node, usually a feature from the visited actions of this state is selected, i.e. it is time to exploit the experiences. Algorithm 2 shows the pseudo code of the *Exploit-based* function.

Algorithm 2: The Exploit-based function

// Add a feature f from the repository of state using *UCB*
// or a new feature using *AOR* ranking
$T_F =$ the number of times the node F is seen
If round$(T_F^b) ==$ round$((T_F + 1)^b)$
  Select f from the repository of this state using *UCB*
Else Select a new feature f, according to max *AOR* value
End if

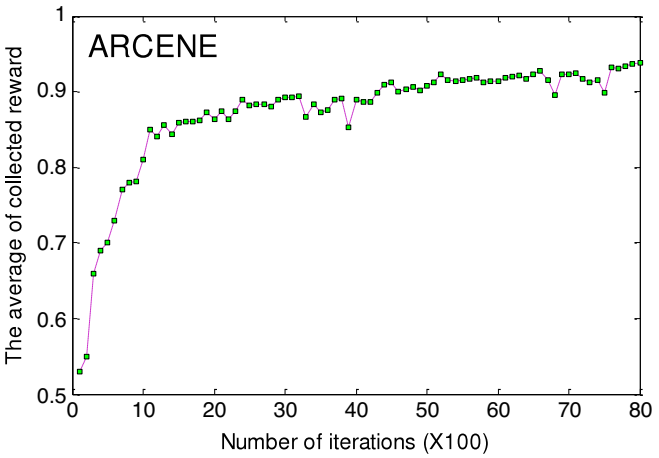

Fig. 2. The average of rewards for the final states of 100 consequent iterations versus the progress of algorithm. This results are on the Arcene dataset.

(3) *The Explore-based function*: On the other hand, when the state has not been seen previously, the *Explore-based* phase must be called and a new feature is selected randomly. Since no action has been selected in this state so far, there are no experiences in this case; therefore it is time to explore the state space freely. Similar to the *Exploit-based* phase, the algorithm proceeds to a new state with this action and returns to the main function to continue its path until the stopping criterion is satisfied. Algorithm 3 indicates this procedure.

Algorithm 3: The Explore-based function

//When we do not have any repository

Select an unseen feature f uniformly at random

4.7. Final selection of features

The values of *AOR* for all features are initially set to zero. In each iteration, some of the features' *AOR* scores are updated. In the beginning, these scores do not indicate their actual merits, and each action is usually randomly selected while the algorithm explores the state space. With the progress of the algorithm, it exploits the experiences. The score updates in the final states in consequent iterations are evidence of this assertion.

Fig. 2 depicts the average of rewards for the final states of 100 consequent iterations along with the progress of the algorithm. It shows both the convergence of the algorithm and that, in the first iterations, when there is a random traverse in the search space, the average reward is low, but afterwards, the promotion of the average reward appears. This diagram demonstrates that, in the initial 100 iterations, the average performance of the learned hypothesis according to the selected features in the final state is almost 53%, which is slightly better compared to the random classification in the binary class datasets. Afterward, at the next iterations, these values grow until they reach stable conditions, and after 8000 iterations this value reaches nearly 92%, which shows the improvement of searching in the most promising regions along with the proper values of *AOR*. This is the condition that aims at the convergence of the algorithm when the values of final states in the consequent iterations are similar. We conduct this experiment on Arcene (Section 5.1.), but the results on other datasets are nearly the same.

After all iterations and traversing the most promising regions, there is a need to use some methods to select the best features of all. To do so, two methods are proposed here.

(1) *F-FSTD (Filter-based FSTD)*: After all iterations, each feature has earned some rewards in some positions demonstrating its importance. These are saved as the *AOR* values of the features and are available in the relevant data structure. Inspired by filtering methods, a procedure is introduced here to select the most significant features. First of all, we sort the features according to the *AOR* in descending mode and then select the top-ranked ones. The major problem with the aforementioned method is that the optimal number of selected features is unknown. In order to solve this challenge, a value has been proposed to limit the number of selected features in a subset [29]:

$$d \approx \sqrt{n}, \tag{9}$$

where d is the upper bound of selected features and n is the total number of features. Eq. (9) states that we must examine the subsets up to this value; however, there is still a dilemma in order to determine the final number of features in the best subset. To overcome this difficulty, we can check the performance of the end learner with some subsets of features using the top-ranked features. More specifically, as the first subset, we only consider top-ranked feature, as the second subset two first

Table 1
UCI benchmark data.

No.	Dataset	Instances	Features
1	Arcene	200	10 000
2	Madelon	2 600	500
3	Spambase	4 601	57
4	Colon	62	2 000
5	Gisette	13 500	5 000
6	Dexter	2 600	20 000

features are considered, and so on. This continues up to a predefined threshold d. The best subset has the best performance of all. In practice, with regard to the diagram of progression, when adding the features proceeds, while there is no increase in the performance, convergence has occurred, and there is no need to proceed further.

(2) *W-FSTD (Wrapper-based FSTD)*: Since the graph of the traversed spaces is available at the end of all iterations, inspired from the wrappers, there is a simple way to select the best subset of features. As a solution, a simple search in the graph space is enough to detect the nodes with the best final reward and introduce them as the best subsets. This is similar to the wrapper approaches which construct several subsets of all features and use the best one. However, the presented approach solves the dilemma of EvE in the wrappers, because the algorithm determines the exploration and exploitation conditions in the state space and select the best subsets regardless of the number of features.

5. Experimental results

This section demonstrates the ability of the *FSTD* algorithm in selecting the best features conducted on several benchmarks. Here, some benchmarks which are used in most of the other papers have been introduced, and then the efficiency and effectiveness of our method is evaluated by comparing *FSTD* with the rival algorithms.

5.1. Selected benchmark datasets

Table 1 indicates the characteristics of UCI benchmark datasets that are used in this paper to validate this method. All of them are used for two-class classification problems with various input variables.

The Arcene dataset is designed to distinguish cancer versus normal patterns; it contains 10 000 continuous input variables as features, including 7000 real features and 3000 random probes. Arcene is constructed from the concatenation of several datasets relevant to related though different concepts (each dataset being related to a particular type of cancer). The disjunction of overlapping concepts is the final concept in the Arcene dataset. The Madelon dataset is a two-class artificial dataset with sparse input variables which is designed to classify random data. This dataset consists of 500 features and the XOR of 5 relevant features is the target concept. Spambase is a dataset with 57 correlated features, and it denotes whether the e-mail was considered spam or not. Colon is a microarray dataset which is considered to be "simple", containing 62 samples and 2000 features.

The task of Gisette dataset is to discriminate between two confusable handwritten digits: the four (4) and the nine (9). This is a two-class classification problem with sparse continuous input variables. We used the version of the database that was prepared for the NIPS 2003 variable and feature selection benchmark by Isabelle Guyon [30]. Dexter is a multivariate dataset with integer attributes. Its task is to filter texts about "corporate acquisitions", and then the dataset is in the realm of text classification.

All datasets are well-known benchmarks that are used several times in FS challenges.

5.2. Experimental settings

FSTD parameters include the values of α and γ when updating the state values according to Eq. (3), that are set to 0.1 and 0.3 respectively. We can set up the update speed of the state values by changing these parameters. To control the exploitation versus the exploration in the Exploit-based function the parameter b is set 0.7 for all trials.

The number of iterations, T, is set in accordance with the size of the state space. For datasets with a large number of features, there is the need to perform several iterations. For example, Arcene with 10 000 features needs 8000 iterations to converge but Spambase, with 57 features, converges after 2000 iterations.

In the stop condition we need to define a suitable value for m, which is the number of times that the values of consecutive states decline. In our experiments on Arcene with 10 000 features and Dexter with 20 000 features, we defined m as 5, which led to proper results. The value of m for the other experiments was set to 3.

For the sake of a fair comparison between this method and the state of the art methods, all FS algorithms were combined with the same end learner, a Gaussian SVM [24].

All reported results are averaged over five independent runs. Each run considers a 5-fold cross-validation (CV) having different distributions. For each fold, *FSTD* is launched with the proper number of iterations that lead to convergence conditions. The runtime was almost 55 min for Dexter, 35 min for Madelon, and less than 10 min for the others.

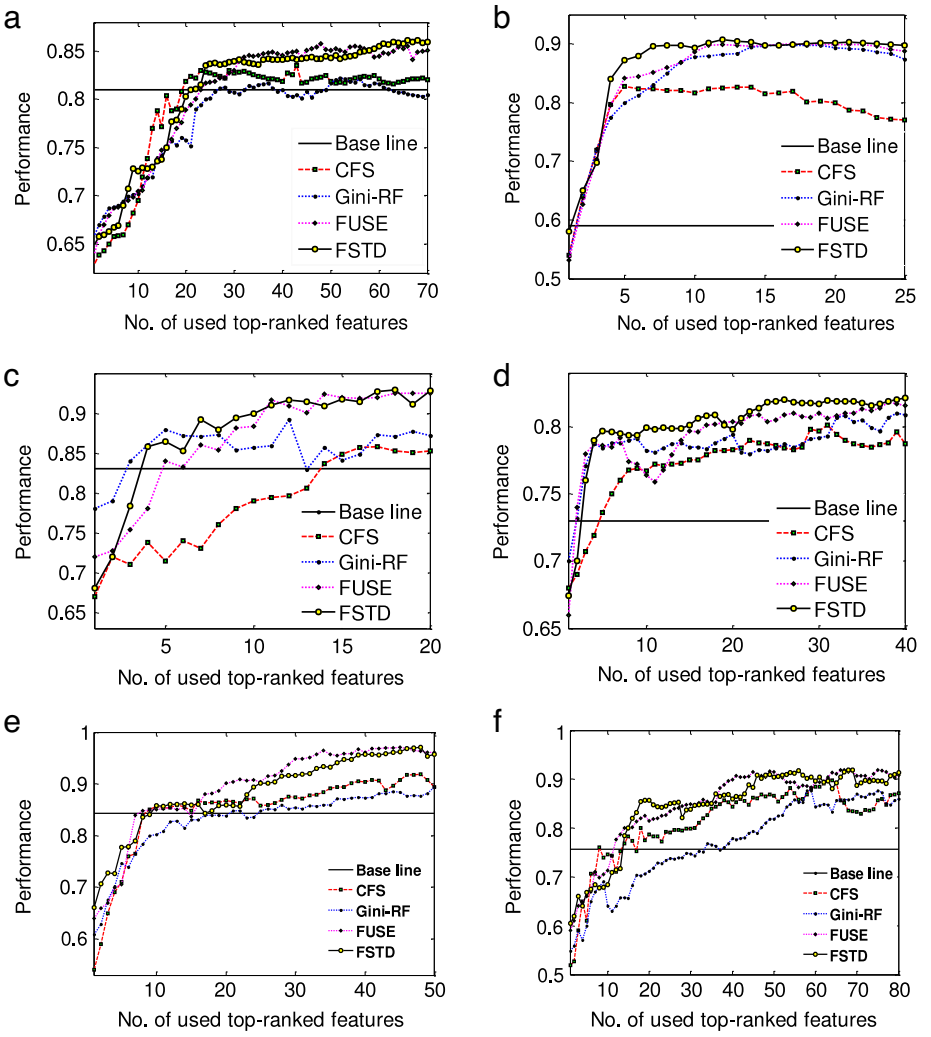

Fig. 3. Performance of *F-FSTD* versus the number of selected features (using top-ranked features) (a) Arcene, (b) Madelon, (c) Spambase, (d) Colon, (e) Gisette, and (f) Dexter.

5.3. Validation of the proposed method

In Fig. 3, the comparison between the performances of *FSTD*, FUSE [7], correlation-based feature selection (CFS) [14] and the random forest based Gini score (Gini-RF) [19] on the presented datasets are shown. The discrete version of FUSE, namely D-FUSE [7], the WEKA implementation of CFS, and the R implementation of Gini-RF with 1000 trees were used.

After selecting the features, the same classifier is considered to calculate the generalization error. The Gaussian SVM [24] is utilized to determine the test error for the top-ranked features in each dataset, because of its robustness and strength. In huge datasets with several features and high interdependency of them, like Arcene, in the absence of any FS method, the generalization error is very high. After CFS, the performance is somewhat better, but it has low performance compared with the others. The other methods have similar results; however, *FSTD* find the proper features sooner than the others (Fig. 3(a)). For Arcene and Colon the growth of performance are surprisingly ascending. Although FUSE and *FSTD* have similar performances, *FSTD* is better in most cases; moreover, it has faster execution time (Fig. 3(b) and (d)). As the Spambase has fewer features than the other datasets, the performance of *FSTD* is soon increased, and the number of iterations to converge the algorithm is low. The reported results are the average of five independent trials of all methods, and in each run a 5-fold cross-validation [1] is used.

The baselines in these diagrams are in accordance with the mentioned predictor in the absence of FS procedures. Obviously, it can be deduced from the depicted diagrams that the influence of FS methods is not only significant in the alleviation of calculations, but, if the proper subset of features is considered, it can improve the performance of the classifier as well. The regions for which the performance of the classifier according to the selected subset of features are above the baseline are evidence of this assertion.

Table 2
The mean performance of classification in accordance with each dataset based on FS methods.

No.	Dataset	Baseline	CFS	Gini-RF	FUSE	F-FSTD
1	Arcene	81.00	79.31	78.15	80.62	80.82
2	Madelon	58.83	78.70	84.23	85.07	86.48
3	Spambase	82.93	78.53	85.48	87.11	87.19
4	Colon	73.06	77.23	78.67	79.33	80.15
5	Gisette	84.29	84.96	82.99	97.46	97.23
6	Dexter	75.69	81.17	76.92	84.82	84.41

Table 3
The best performance of classification in accordance with each dataset based on FS methods.

No.	Dataset	Baseline	CFS	Gini-RF	FUSE	F-FSTD
1	Arcene	81.00	83.51	82.24	85.83	86.12
2	Madelon	58.83	82.81	89.91	90.09	90.89
3	Spambase	82.93	85.81	89.27	92.59	92.83
4	Colon	73.06	80.12	81.01	81.79	82.11
5	Gisette	84.29	91.94	88.74	91.88	91.88
6	Dexter	75.69	89.84	89.62	91.88	91.88

Some preprocessing steps have been used on the datasets before the FS stages are reached. With regard to the perturbation of the features in Madelon and Colon, to prevent the masking of some features through the others, there was the need to normalize all features in the same range. So Madelon and Colon were centered and normalized. Moreover, Madelon, which has a large number of instances, was very costly in the evaluation stages, so the procedure of instance reduction was used to reach an acceptable condition. We lessen the number of instances to 10% of all, as an efficient subsample only for the evaluation phase.

As can be seen in the diagrams, after feature selection the first gain is the growth of performance. The datasets contain several irrelevant features, and moreover progress in the classification rate the time and space complexity of the problem are decreased remarkably. Table 2 shows the mean values of performances of each method on each dataset according to the selected features.

The proposed method with the *FUSE* algorithm demonstrates better results compared with the other algorithms on all datasets. For the first four datasets in Table 2, the mean performance after *F-FSTD* is the foremost, but for the last two datasets *FUSE* is forerunner.

On the Arcene dataset, where all of the samples are considered, after 8000 iterations, the convergence condition has appeared and 70 top-ranked features according to the *AOR* values are selected. The average performance in this case is close to the baseline; in the best case, the performance without applying any FS method is 81%, but after using the *FSTD* method this value rises to 86.12% for the best subset.

In the Colon dataset, which has several irrelevant features, the error rate before FS is low; but after selecting best features this value declines. After selecting 40 features in this case, based on *F-FSTD* the best performance was 82.11%, versus 73.06% with the lack of FS. For FUSE, CFS, and Gini-RF, the values are 81.79%, 80.12%, and 81.01%, respectively.

Madelon is also a dataset with many improper features that has a performance near random classification before using FS, i.e., 58.83%. This value increases by more than 32 percentage points after FS; it is 90.89% for *F-FSTD*.

In Table 3, each FS method's best performance on all datasets is represented.

The filter-based method of *FSTD*, namely *F-FSTD*, was employed to select the best features. In the wrapper-based style, *W-FSTD*, the best node of the graph is selected after comparison. Eventually, the selected subset of features is not necessarily repetitive in most features, as in the filter mode. Moreover, the final number of features in the best subset can be determined automatically from the lattice of the subset of features.

Fig. 4 demonstrates the performance of this method on the Arcene dataset. Whereas the dataset has several irrelevant features, after selecting the best features the best performance is almost 87%. As the subsets are selected from several regions, when we want to take some samples, we are able to prepare features with low cost of construction. In the applicable problems it is of great importance to know the appropriate subset of features with a low cost of preparation.

The subset of features contain up to 78 features in total. The results on the other datasets are the same.

6. Conclusion and future work

In most datasets, some features are important for a specific problem, but for a given task only a small subset of features is relevant, and, for a specific job, there is more emphasis on one aspect than on others. In this research, in order to select the best subset of features, we consider the problem as an MDP and introduce an innovative method to traverse the environment. Then, any subset of features is considered as a state. By selecting a feature, the state will be changed to a new one. We must select the best action in each state that leads to gathering the maximum rewards in an episode. Each episode starts from an empty subset, and selecting a feature changes the current state to another. This procedure continues until a stop criterion

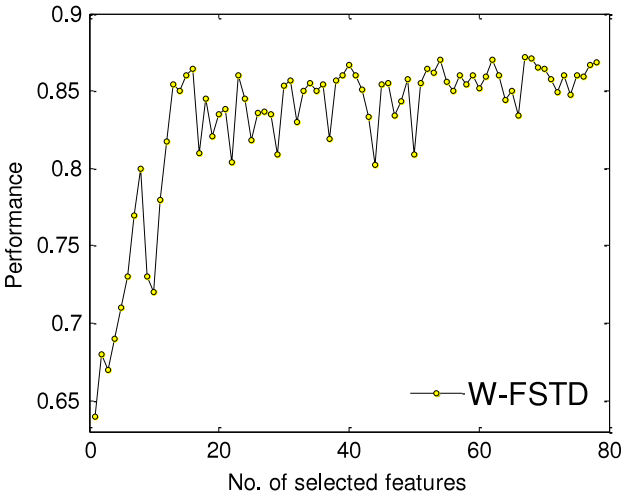

Fig. 4. Performance versus the number of selected features in *W-FSTD* style on the Arcene dataset. We consider the search graph and select the best subset of features in each size.

is satisfied, and this condition could be deterioration of the value of new state compared with the value of current state. In this work, an SVM is regarded as the evaluation function with the features of a specific state. To reduce the complexity of classification, an instance reduction is considered as a preprocessing stage.

The importance of each feature is determined with a proposed criterion, namely the average of rewards, *AOR*. Taking inspiration from the Shapley value method, the *AOR* is defined as the average of rewards which a feature has gained so far. The difference between the value of a state and its successor defines the reward of a feature that causes this transform in this iteration. The search in the state space results in traversing an extensive search tree, but, by controlling the repetitive nodes, the environment could be considered as a search graph, and we can utilize the recent information about the subset of features. Given that the RL method is iterative, in the first iterations of the algorithm, it explores in the state space, while it exploits the gathered experiences with the progress of iterations.

Finally, the best features could be selected from the top-ranked ones according to the *AOR*, like filter methods, i.e. *F-FSTD*, or from the best subset of the search graph inspired from the wrappers, i.e. *W-FSTD*. In the problems aimed at determining the importance of features, the presented method results in a proper feature weighting. The main object of this study is compatibility with all conditions, and the results confirm this assertion. In particular, this is a suitable method for datasets with a large number of features.

Further work in this field includes changing the stopping criteria conditions. As an idea inspired from the method of search Plus-*l* Minus-*r*, we can add the best feature to a state and go to another state. Moreover, sometimes we can eliminate the worst feature from a state and move to another one. As other work we can replace the other classifier and extend the research to multiclass datasets. Moreover, instead of defining a certain number of iterations as an input parameter, we can detect the convergence condition of the proposed algorithm adaptively according to all datasets.

Acknowledgment

This work is supported by Iranian Telecommunication Research Center (ITRC) under Grant No. T/500/13226.

References

[1] R. Gutierrez-Osuna, Introduction to Pattern Analysis, Texas A&M University, 2002.
[2] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Journal of Machine Learning Research (JMLR) (2003) 1157–1182.
[3] A.G. Sutton, S. Richard, Reinforcement Learning, An Introduction, MIT Press, Cambridge, MA, 1998.
[4] L. Kocsis, C. Szepesv'ari, Bandit based Monte Carlo planning, in: ECML, 2006, pp. 282–293.
[5] S.M. Hazrati, A. Hamzeh, S. Hashemi, A game theoretic framework for feature selection, in: FSKD, 2012, pp. 845–850.
[6] T.M. Mitchell, Machine Learning, McGraw-Hill Science/Engineering/Math, 1997. March 1.
[7] R. Gaudel, M. Sebag, Feature selection as a one-player game, in: ICML, Haifa, Israel, 2010, pp. 359–366.
[8] R. Kohavi, G. John, Wrappers for feature selection, Artificial Intelligence (1997) 273–324.
[9] W. Duch, K. Grabczewski, T. Winiarski, J. Biesiada, A. Kachel, Feature selection based on information theory, consistency and separability indices, in: International Conference on Neural Information Processing, ICONIP '02, 2002.
[10] E.S. Pearson, H.O. Hartley, Biometrika Tables for Statisticians, Vol. II, CUP, 1972.
[11] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, second ed., John Wiley & Sons, USA, 2001.
[12] W.T. Eadie, F.E. Drijard, M. Roos, B. Sadoulet, Statistical Methods in Experimental Physics, North-Holland, Amsterdam, 1971, pp. 269–271.
[13] K. Kira, L.A. Rendell, A practical approach to feature selection, Machine Learning (1992) 249–256.
[14] M.A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in: ICML'00, 2000, pp. 359–366.
[15] M. Boull'e, Compression-based averaging of selective NaiveBayes classifiers, in: JMLR'08, pp. 1659–1685.
[16] T. Zhang, Adaptive forward–backward greedy algorithm for sparse learning with linear models, in: NIPS'08. pp. 1921–1928.

[17] D. Margaritis, Toward provably correct feature selection in arbitrary domains, in: NIPS'09, pp. 1240–1248.
[18] R. Tibshirani, Regression shrinkage and selection via the Lasso, Journal of the Royal Statistical Society. Series B (1994) 267–288.
[19] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth and Brooks, 1984.
[20] P. Rolet, M. Sebag, O. Teytaud, Boosting active learning to optimality, a tractable Monte Carlo, Billiard-based algorithm, in: ECML'09. pp. 302–317.
[21] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed Bandit problem, Machine Learning (2002) 235–256.
[22] A.P. Flach, The geometry of ROC space, understanding machine learning metrics through ROC isometrics, in: ICML'03.
[23] W.T. Saul, Teukolsky, Numerical Recipes in C, The Art of Scientific Computing, Cambridge University Press, Cambridge, 1992.
[24] R. Collobert, S. Bengo, J. Marithoz, Torch, a modular machine learning software library, Technical report, IDIAP, 2002.
[25] G.L. Ritter, H.B. Woodruff, S.R. Lowry, T.L. Isenhour, An algorithm for a selective nearest neighbor decision rule, Department of Chemistry, University of North Carolina, Chapel Hill, 1975, pp. 665–669.
[26] D.L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man and Cybernetics (1972) 408–421.
[27] L.S. Shapley, A value for n-person games, in: H.W. Kuhn, A.W. Tucker (Eds.), Contributions to the Theory of Games, in: Annals of Mathematics Studies 28, vol. II, Princeton University Press, Princeton, 1953, pp. 307–317.
[28] R. Coulom, Eddicient, selectivity and backup operators in Monte Carlo tree search, Computers and Games (2006) 72–83.
[29] L. Breiman, Random forests, Machine Learning (2001) Statistics Department, University of California, Berkeley.
[30] I. Guyon, S.R. Gunn, A. Ben-Hur, G. Dror, Result analysis of the NIPS 2003 Feature Selection challenge, in: NIPS'04, 2004, pp. 545–552.