Reinforced variable selection

Yuan Le, Yang Bai & Fan Zhou

STATISTICAL THEORY AND RELATED FIELDS
https://doi.org/10.1080/24754269.2025.2516346

Reinforced variable selection

Yuan Le[a], Yang Bai[b] and Fan Zhou[b]

[a]School of Mathematics and Statistics, Fuzhou University, Fuzhou, People's Republic of China; [b]School of Statistics and Data Science, Shanghai University of Finance and Economics, Shanghai, People's Republic of China

ABSTRACT
Variable selection identifies the best subset of covariates when building the prediction model, among all possible subsets. In this paper, we propose a novel reinforced variable selection method, called 'Actor-Critic-Predictor'. The actor takes an action to choose variables and the predictor evaluates the action based on a well-designed reward function, where the critic learns the reward baseline. We model the variable selection process as a multi-armed bandit and update the subset of selected variables using a natural policy gradient algorithm. We provide an analytical framework on how different errors impact the performance of our method theoretically. Large amounts of experiments on synthetic and real datasets show that the proposed framework is easily implemented and outperforms classical variable selection methods in a wide range of scenarios.

ARTICLE HISTORY
Received 11 November 2024
Revised 13 March 2025
Accepted 1 June 2025

KEYWORDS
Variable selection; reinforcement learning; natural policy gradient

1. Introduction

Variable selection (VS), also known as feature selection, is one of the fundamental problems in statistics and machine learning. Variable selection has been widely used in various fields, including causal inference (Shortreed & Ertefaie, 2017), computer vision (Barbu et al., 2016), nature language processing (Lewis, 1992), and recommender systems (Mirzadeh et al., 2005). One typical example is the genome-wide association study (GWAS) (Guyon et al., 2002). A standard GWAS approach is to scan tens of thousands of genes and look for genetic markers that can be used to predict the presence of a disease. A small subset of genes usually contains most of the disease information. In this case, variable selection is required to eliminate unrelated genes and improve the classification performance. In general, an effective variable selection helps reduce input dimensionality, improves prediction performance, and facilitates data understanding.

VS methods can be divided into three main categories according to the ways they select variables. Filtering methods independently rank all variables based on some predefined score functions. RELIEF (Kira & Rendell, 1992) measures the variable impact on the neighbourhood of each training sample. The neighbourhood is determined by using the Euclidean distance built with all candidate variables. Sure independence screening (SIS) (Fan & Lv, 2008) ranks features according to some marginal utility, for example, the marginal correlation when fitting a linear regression. Some other score functions include statistical

CONTACT Yang Bai statbyang@mail.shufe.edu.cn School of Statistics and Data Science, Shanghai University of Finance and Economics, 777 Guoding Road, Yangpu District, Shanghai 200433, People's Republic of China

scores (variance, t-score, chi-squared score, Gini index Gini, 1912), similarity scores (Laplacian score He et al., 2005, SPEC Zhao & Liu, 2007, Fisher score Hart et al., 2000, Trace Ratio Nie et al., 2008), and information-theoretical-based scores (Mutual Information Battiti, 1994, MRMR Peng et al., 2005, CI Lin & Tang, 2006, JMI Yang & Moody, 1999, CMI Vidal-Naquet & Ullman, 2003). Despite the computation efficiency (Fard et al., 2013), the variables selected by filtering methods are non-optimal since the filtering is done in the preprocessing step and is independent of the main task.

Wrapper methods evaluate the power set generated by all the candidate variables and estimate the generalization error for each possible subset. Although these methods are theoretically optimal if the computing resources are rich and the memory storage is large enough, one big challenge in practice is how to effectively explore all possible subsets of variables and minimize the computational cost. Some greedy (forward or backward) searching methods (Hall, 2000) are easy to implement but may end up with some local optimums. Some heuristic methods are proposed to address the 'Exploration versus Exploitation' (EvE) dilemma, such as mixing forward and backward selection (T. Zhang, 2011), combining global and local search (Boullé, 2007).

Embedded methods assume that only a small subset of variables is useful and thus apply regularization in model training to ensure sparsity. Tibshirani (1996) introduces Least absolute shrinkage and selection operator (Lasso) to do variable selection where the L_1 penalty is used to fit the regression model. Some other penalty functions may also be considered, such as smoothly clipped absolute deviation (SCAD) (Fan & Li, 2001) and minimax concave penalty (MCP) (C. Zhang, 2010). When it comes to multi-classification problems or multi-objective regression problems, $L_{p,q}$ regularization is used, and $L_{2,1}$ norm is commonly used. Nie et al. (2010) proposes a computationally efficient and robust algorithm by applying the $L_{2,1}$ norm to both the loss function as well as the regularization term. As for VS in unsupervised learning, Cai et al. (2010) proposes multi-cluster variable selection by first obtaining pseudo-labels from the graph Laplacian through spectral analysis and then using the conventional regularization methods in supervised learning. However, most of these methods are designed for linear model structure. Bach (2008) extends the regularization method to the nonlinear case by multiple kernel learning, and more about kernel methods in VS (Chen et al., 2017; Varma & Babu, 2009) have been studied. Another way to deal with nonlinear problems is through tree models. A well-known method is to use random forest (Breiman, 2001) to calculate the importance score of each variable. Recently, Lemhadri et al. (2021) introduces regularization to neural networks and proposes LassoNet to achieve feature sparisity. Embedded methods also struggle with the EvE dilemma as the wrapper methods do, although they usually search in a constrained model space.

How to define a valid hypothesis space and introduce an efficient searching strategy is a fundamental problem for variable selection. Some recent studies try to address this issue by using reinforcement learning (RL). As a 'trial-and-error' method, RL can gradually approach the optimal subset of variables by learning from historical trajectories. Gaudel and Sebag (2010) formalizes the variable selection problem as a Markov decision process (MDP) and proposes an RL-based searching algorithm called feature UCT selection (FUSE) together with an exploration strategy (upper confidence tree, UCT). FSTD (Fard et al., 2013) modifies FUSE by using Upper Confidence Gragh (UCG) (Fard et al., 2012) instead of UCT. FSTD also replaces the original Monte Carlo method in FUSE by temporal difference (TD) method (Sutton, 1988), which is further improved by Fang et al. (2019) with deep Q-network (DQN). K. Liu et al. (2019) proposes a multi-agent RL framework where each variable is regarded as an agent and the state space is the characteristics of the selected variables. All these methods are value-based, and can only learn deterministic policies. They also require some tree-based searching strategies to ensure the exploration effects and approximate the optimal policy. Moreover, the stopping criteria of these methods needs to be carefully designed, which makes the whole training process quite complicated. Recently, Y. Liu and Ročková (2023) uses Thompson sampling to do variable selection from Bayesian perspective and gives some theoretical results on regret bounds.

Considering the disadvantages of existing RL methods, this paper proposes a novel variable selection algorithm called 'Actor-Critic-Predictor' (ACP). The actor takes an action to choose variables. The predictor evaluates the action by giving a reward, helping the actor update its policy, and guiding the critic to learn a baseline of the reward. We use a policy-based method to discover the optimal policy, where the actor can adaptively change the searching scope. We opt for a policy-based approach over value-based methods due to several key reasons. Firstly, policy-based methods with stochastic policies naturally balance exploration and exploitation without requiring additional mechanisms like ϵ-greedy, which is crucial when navigating the exponentially growing search space of variable selection. Furthermore, stochastic policies provide a natural representation of uncertainty, where selection probabilities reflect our confidence in each variable's importance, offering more nuanced information than binary decisions. From an optimization perspective, policy-based methods enable direct gradient-based optimization, which proves more efficient and stable than the tree-based search strategies employed by value-based methods such as FUSE and FSTD. Experiments on both synthetic and real datasets show that this framework is general and easily implemented. The whole architecture of the proposed ACP method is illustrated in Figure 1. Our main contributions are summarized as follows.

- We pay attention to stochastic policies and firstly develop a policy-based RL algorithm using natural policy gradient to solve variable selection problems.
- We theoretically analyze the impact of both the optimization error and the model error on the final performance.
- Some well-designed numerical experiments in both regressions and classifications provide some insights into the selection consistency which are ignored by previous RL-based methods. Extensive experiments on both synthetic and real datasets show the superiority of the proposed algorithm over some classical VS methods.

2. Reinforced variable selection

We consider a general variable selection problem as follows,

$$y = f(\mathbf{x}_{\text{true}}) + \epsilon, \tag{1}$$

where $\mathbf{x}_{\text{true}} = (x_1, x_2, \ldots, x_d)^\top \in \mathbb{R}^d$ consists of all the true covariates that are related to the response y. f denotes some unknown function, and ϵ represents a random noise which is independent of $\mathbf{x}_{\text{true}}$. In many cases, we only observe $\mathbf{x} = (x_1, \ldots, x_d, \ldots, x_p)^\top \in \mathbb{R}^p$ other than $\mathbf{x}_{\text{true}}$. $\mathbf{x}$ contains all elements of $\mathbf{x}_{\text{true}}$, although some unrelated covariates are also included. The main goal of variable selection is to find all the true covariates in $\mathbf{x}_{\text{true}}$ given $\mathbf{x}$ and y.

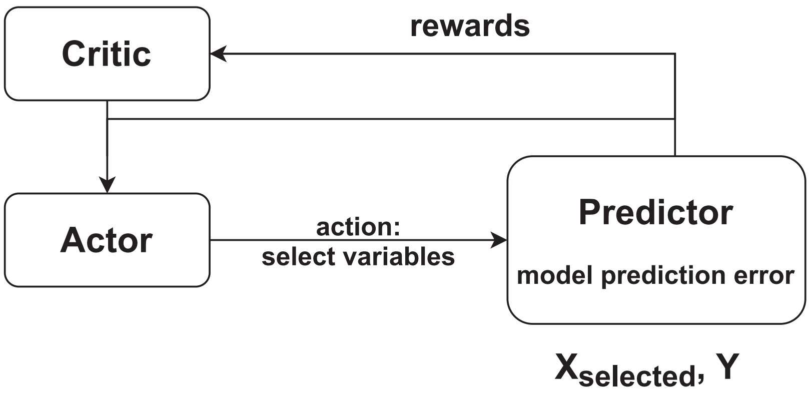

Figure 1. Reinforcement learning framework for variable selection.

2.1. Variable selection as multi-armed bandit

We formalize the variable selection problem as a multi-armed bandit (MAB), which needs to determine the action set $\mathcal{A}$ and the corresponding reward function $\mathbf{r}$. For variable selection, $\mathcal{A} = \{0, 1\}^p$ and $\mathbf{a} = (a_1, \dots, a_p)^\top \in \mathcal{A}$ is any action to take. In particular, $a_j = 1$ for $j \in \{1, 2, \dots, p\}$ if the jth variable is selected. $\mathbf{x}^\mathbf{a} := \{x_j : a_j = 1\}_{j=1}^p$ denotes the set of covariates selected by action $\mathbf{a}$.

Our goal is to find a variable subset to minimize the generalization error. Naturally, we let $\mathbf{r} = \{r^\mathbf{a}\}_{\mathbf{a}\in\mathcal{A}}$ and define the reward of a specific action $\mathbf{a}$ as the negative value of the generalization error given as follows,

$$r^\mathbf{a} = -\mathbb{E}_{\mathbf{x},y}\left[l(y, f_\mathbf{a}(\mathbf{x}^\mathbf{a}))|\mathbf{a}\right], \tag{2}$$

where l is some pre-defined loss function, and $f_\mathbf{a}$ is the best predictor of y using $\mathbf{x}^\mathbf{a}$ such that $f_\mathbf{a} = \arg\min_f \mathbb{E}_{\mathbf{x},y}[l(y, f(\mathbf{x}^\mathbf{a}))|\mathbf{a}]$. The reason we use generalization error instead of the fitting error to represent the reward function is that the policy tends to select all the m variables to minimize the fitting error. MAB can be viewed as a one-step MDP, i.e., in the six tuple $(\mathcal{S}, \mathcal{A}, \mathbf{P}, \mathbf{r}, \gamma, \rho)$ of an MDP, where the state set $\mathcal{S}$ has only one element, the initial state distribution ρ is a Dirac distribution of that element, there is no state transfer probability $\mathbf{P}$, and the discount factor $\gamma = 0$.

2.2. Policy

A policy is a distribution of $\mathbf{a} \in \mathcal{A}$, denoted by π. Let r_t be the reward at timestep t which satisfies $\mathbb{E}[r_t|\mathbf{a}] = r^\mathbf{a}$, and define

$$\rho(\pi) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[r_1 + r_2 + \cdots + r_t + \cdots + r_T|\pi\right] = \sum_{\mathbf{a}\in\mathcal{A}} \pi(\mathbf{a}) r^\mathbf{a}. \tag{3}$$

Our goal is to find a policy π to maximize $\rho(\pi)$.

Let $\mathbf{a}^*$ be the action that selects all elements in $\mathbf{x}_{\text{true}}$ without any unrelated covariate being included. Assuming that $r^{\mathbf{a}^*}$ is the largest of all rewards, then theoretically there exists an optimal policy $\pi^* = \arg\max_\pi \rho(\pi)$ of the following form:

$$\pi^*(\mathbf{a}) = \begin{cases} 1, & \text{if } \mathbf{a} = \mathbf{a}^*, \\ 0, & \text{otherwise.} \end{cases}$$

Suppose that each variable has a probability of being selected, denoted by θ_j where $j = 1, \dots, p$. Then we can define such a stochastic policy as

$$\pi_{\boldsymbol{\theta}}(\mathbf{a}) = \prod_{j=1}^{p} \left\{ \theta_j^{a_j} (1 - \theta_j)^{1-a_j} \right\},$$

where $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)$ and satisfies $\sum_{\mathbf{a} \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(\mathbf{a}) = 1$. Each a_j in this policy is obtained by sampling from a Bernoulli distribution parameterized by θ_j. The larger θ_j is, the more likely the jth variable will be selected. We want the probabilities corresponding to the true variables to be as large as possible after training, while the probabilities of the irrelevant variables are not very large.

3. The Actor-Critic-Predictor algorithm

In this section, we introduce the proposed Actor-Critic-Predictor (ACP) algorithm and all its important components. We also analyze the impact of different errors on the final performance from theoretical perspective.

We resort to natural policy gradient when optimizing the objective function defined in (3). According to Sutton and Barto (2018), we have

$$\nabla_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}) = \sum_{\mathbf{a}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\mathbf{a}) r^{\mathbf{a}} = \mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}} \left[(r_t - r_{\text{baseline}}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t) \right], \tag{4}$$

where $\nabla_{\boldsymbol{\theta}}$ denotes the derivative with respect to $\boldsymbol{\theta}$. r_{baseline} is the reward baseline, which is uncorrelated to the action. a_t and r_t are the action, and reward at the tth timestep. By calculation, it is easy to get $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) = \frac{\mathbf{a}}{\boldsymbol{\theta}} - \frac{1-\mathbf{a}}{1-\boldsymbol{\theta}}$. With the gradient, we can update $\boldsymbol{\theta}$ by gradient ascent. However, the direction of the gradient is not the most rapid ascent direction. Hence, Kakade (2001) proposes the natural policy gradient method:

$$\boldsymbol{\theta}^{(m+1)} = \boldsymbol{\theta}^{(m)} + F(\boldsymbol{\theta}^{(m)})^\dagger \nabla_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}^{(m)}),$$

where $F(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{a} \sim \pi_{\boldsymbol{\theta}}} [\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a})^\top]$. $F^\dagger$ is the Moore-Penrose pseudoinverse of F and m is the training step. It is difficult to compute $F(\boldsymbol{\theta})^\dagger$ when the dimension is very high. So we consider computing $F(\boldsymbol{\theta})^\dagger \nabla_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta})$. Define

$$L_\pi^{\boldsymbol{\theta}}(\mathbf{w}) = \mathbb{E}_{\mathbf{a} \sim \pi} \left[(\mathbf{w}^\top \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) - (r^{\mathbf{a}} - r_{\text{baseline}}))^2 \right], \quad \mathbf{w} \in \mathbb{R}^m. \tag{5}$$

Let $\mathbf{w}_{\boldsymbol{\theta}}^*$ denote the smallest norm solution to minimize $L_\pi^{\boldsymbol{\theta}}(\mathbf{w})$, and then we have

$$\mathbf{w}_{\boldsymbol{\theta}}^* = F(\boldsymbol{\theta})^\dagger \nabla_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}).$$

Therefore, the updates of ACP algorithm are

$$\boldsymbol{\theta}^{(m+1)} = \boldsymbol{\theta}^{(m)} + \mathbf{w}_{\boldsymbol{\theta}^{(m)}}^*, \quad \mathbf{w}_{\boldsymbol{\theta}^{(m)}}^* = \arg\min_{\|\mathbf{w}\|_2 \le W} L_\pi^{\boldsymbol{\theta}^{(m)}}(\mathbf{w}). \tag{6}$$

Note that the iterations above are in the population level. In the implementation of the algorithm, expectations are approximated by the empirical means of samples. Besides, the real reward $r^{\mathbf{a}}$ is not known, and we only have its estimate $\hat{r}^{\mathbf{a}}$ from the predictor by solving a supervised learning problem. Therefore, the difference between $\mathbf{r}$ and $\hat{\mathbf{r}}$ may induce an empirical gap.

We make some modifications to the notation of the objective function by adding a subscript $\mathbf{r}$ to denote the objective function $\rho_{\mathbf{r}}(\pi)$ obtained using the specified reward function $\mathbf{r}$ and under the policy π. Assuming that $\hat{\pi}$ is the policy obtained by ACP algorithm, we can theoretically evaluate the performance of $\hat{\pi}$ by measuring the difference between the objective function of the optimal policy and itself, i.e., $\rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\mathbf{r}}(\pi^*)$. The following proposition shows that this performance discrepancy can be decomposed into two parts: optimization error and reward modelling error.

Proposition 3.1: *Let $\hat{\pi}^* = \arg\max_\pi \rho_{\hat{\mathbf{r}}}(\pi)$ denote the theoretically optimal solution where $\hat{\mathbf{r}}$ is the estimate of $\mathbf{r}$, and then the difference between $\hat{\pi}$ and $\hat{\pi}^*$ is the optimization error, denoted as ϵ_{opt}. Total variation difference $D_{\text{TV}}(\hat{\pi} \| \hat{\pi}^*) = \sup_{\mathbf{a}} |\hat{\pi}(\mathbf{a}) - \hat{\pi}^*(\mathbf{a})|$ is used to measure this error. The difference between $\hat{\mathbf{r}}$ and $\mathbf{r}$ is the reward modelling error. Assume $|\hat{r}^{\mathbf{a}} - r^{\mathbf{a}}| \le \epsilon_{\text{mod}}$ for all $\mathbf{a} \in \mathcal{A}$, and further assume $||\mathbf{r}||_\infty \le R$ where R is a given constant. Then*

$$\left|\rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\mathbf{r}}(\pi^*)\right| \le 2R\epsilon_{\text{opt}} + 2\epsilon_{\text{mod}}.$$

The proof of this proposition can be found in Appendix 1.

Remark 3.1:
- The assumption that $||\mathbf{r}||_\infty \le R$ ensures that the rewards are bounded, which is typically satisfied in practice through proper normalization.
- The assumption that $|\hat{r}^{\mathbf{a}} - r^{\mathbf{a}}| \le \epsilon_{\text{mod}}$ for all $\mathbf{a} \in \mathcal{A}$ requires that our reward estimator has bounded error across all possible variable subsets. This is reasonable when the predictor is well-trained and the reward function is well-behaved. If the true reward function is linear, and we use Lasso as the predcitor. Under the restricted eigenvalue condition, we can achieve a convergence rate of $O(d\frac{\log(p)}{n})$ for d-sparse signals in p dimensions, with sample size n (Bickel et al., 2009). If we use neural network as the predictor, for functions with bounded second derivatives, $\epsilon_{\text{mod}} \le O(s^{-2/p})$ where s is the number of parameters and p is the input dimension (Yarotsky, 2017).
- The optimization error ϵ_{opt}: Agarwal et al. (2021) provides a theoretical analysis on natural policy gradient algorithm. Under their assumptions, our algorithm can achieve $\epsilon_{\text{opt}} \le O(\sqrt{\frac{\log|\mathcal{A}|}{M}})$, where $\mathcal{A}$ is the action set, M is the iterations that the algorithm runs for. In our case, $|\mathcal{A}| = 2^p$, and therefore, $\epsilon_{\text{opt}} \le O(\sqrt{\frac{p}{M}})$.

3.1. Algorithm implementation

We now introduce the main procedure of the proposed ACP algorithm, where more details can be found in Appendix 2.

Now we have n observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the expectation in (2) should be replaced by the empirical mean and $f_{\mathbf{a}}$ needs to be estimated. To avoid over-fittings, the samples used to estimate $f_{\mathbf{a}}$ should be independent of those used to compute the empirical mean of the loss function.

To address this problem, we employ a data splitting strategy. After getting an action $\mathbf{a}_t$ at timestep t, the whole dataset is randomly split into two parts, where the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n_1}$ and validation set $\{(\mathbf{x}_j, y_j)\}_{j=1}^{n_2}$ have n_1 and n_2 samples respectively, such that $n_1 + n_2 = n$. We use the training set to build the predictor $\hat{f}_{\mathbf{a}}$ and the validation set to compute the reward, i.e.,

$$\hat{f}_{\mathbf{a}_t} = \arg\min_{f} \frac{1}{n_1} \sum_{i=1}^{n_1} \left[l(y_i, f(\mathbf{x}_i^{\mathbf{a}_t})) \right], \tag{7}$$

$$r_t = -\frac{1}{n_2} \sum_{j=1}^{n_2} l(y_j, \hat{f}_{\mathbf{a}_t}(\mathbf{x}_j^{\mathbf{a}_t})). \tag{8}$$

Thus, we can guarantee that $\mathbb{E}[r_t|\mathbf{a}] = \mathbb{E}[l(y, \hat{f}_{\mathbf{a}}(\mathbf{x}^{\mathbf{a}}))|\mathbf{a}] = \hat{r}^{\mathbf{a}}$. The procedure for computing the reward function is summarized in Algorithm 1 in Appendix 2.

The next step is to calculate $\mathbf{w}$. Note that (5) can be viewed as a linear regression problem with $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a})$ as the covariates and $\hat{r}^{\mathbf{a}} - r_{\text{baseline}}$ as the response. Therefore, a set of actions $\mathbf{a}_{\text{batch}} = \{\mathbf{a}_1, \ldots, \mathbf{a}_B\}$ with batch B can be sampled by the policy $\pi_{\boldsymbol{\theta}}$, and then $\hat{\mathbf{r}}_{\text{batch}} = \{\hat{r}_1, \ldots, \hat{r}_B\}$ and $\nabla_{\text{batch}} = \{\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_1), \ldots, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_B)\}$ can be computed by the aforementioned procedures to calculate the reward. We employ a simple critic to learn the baseline r_{baseline}, which is calculated as the exponentially weighted moving average of the observed rewards. Finally, $\mathbf{w}$ is obtained by a linear regression with ∇_{batch} as covariates and $\hat{\mathbf{r}}_{\text{batch}} - r_{\text{baseline}}$ as the response. To avoid the case of ordinary least squares regression with no solution for $B < p$, we use a ridge regression with a small penalty parameter to ensure that the solution is feasible. In addition, to ensure that the denominator of $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a})$ is not 0, θ must be strictly greater than 0 and less than 1. We clip θ to $[e, 1-e]$ to satisfy this constraint, where e is a small probability quantity. The pseudo code of ACP algorithm is summarized in Algorithm 2 in Appendix 2.

3.2. Predictor for reward

According to Proposition 3.1, a good estimate of the reward function is strikingly important since it impacts the final performance through the model error. A simple and direct way is to fit a linear model, but there will be a problem of model misspecification. Intuitively, if the model is wrong, the reward computed based on this model can not well represent the value of the selected variable subset. We will illustrate it with synthetic data later. To address this problem, we use neural networks (NNs) to model the predictor. With the powerful representation capabilities of NNs, we can fit the real function well. The network architecture of the predictor and more training details are in Appendix 2.

Training a deep learning model is usually time consuming in practice. To accelerate the training process, we create a dictionary to store all the generated action-reward pairs. The keys index the subsets of selected variables and the values are the corresponding prediction error. Each time we encounter a same set of variables that has already been memorized by the dictionary, we can directly locate the subset and use the stored value to represent the reward function instead of recalculating it. If the set of variables is unobserved before, we create a new index for it and compute the corresponding reward. With this acceleration strategy, the

Table 1. List of true fuctions.

True f	Details
1. $y = \sum_{j=1}^{8} \beta_j x_j$	$x_j \overset{\text{i.i.d.}}{\sim} N(0,1)$ for $j = 1, \ldots, p$.
2. $y = \sum_{j=1}^{8} \beta_j x_j$	$\{x_j\}_{j=1}^{p} \sim N(\mathbf{0}, \boldsymbol{\Sigma})$, $\boldsymbol{\Sigma}_{ij} = 0.9^{\lvert i-j\rvert}$.
3. $y = \beta_1 x_1 + \beta_2 x_2 x_3 + \beta_3 x_4 + \beta_4 x_5 x_6 + \beta_5 x_7 + \beta_6 x_8$	$x_j \overset{\text{i.i.d.}}{\sim} N(0,1)$ for $j = 1, \ldots, p$.
4. $y = \mathbf{W_2}^{\top} \text{relu}(\mathbf{W_1 X})$	$\mathbf{X} \sim N(\mathbf{0}, \boldsymbol{I}_8), \mathbf{W_1} \in \mathbb{R}^{32\times 8}, \mathbf{W_2} \in \mathbb{R}^{32}$.
5. $P(y = 1) = 1/(1 + \exp(-\sum_{j=1}^{8} \beta_j x_j))$	$x_j \overset{\text{i.i.d.}}{\sim} N(0,1)$ for $j = 1, \ldots, p$.
6. Multi-classification (5 classes)	`make_classification` (n_samples = 200, n_features = 50, n_informative = 8, n_redundant = 0, n_repeated = 0, n_classes = 5) in scikit-learn.

training time can be significantly reduced by about 60 percent as many repeated calculations are skipped.

4. Numerical experiments

In this section, we conduct several numerical experiments to examine the empirical performance of the proposed ACP algorithm under different model assumptions, including linear regressions, non-linear regressions and classifications. For the synthetic experiments, the default number of true covariates is $d = 8$, the default number of candidate covariates is $p = 50$, and the default sample size is $n = 200$. The random noise ϵ follows a normal distribution, such that $\epsilon \sim N(0, 0.5^2)$. The details of the data generating procedure are summarized in Table 1. In all situations, we assume that the coefficients β's of the true covariates have the same value 1, i.e., $\beta_j = 1$ for all x_j in $\mathbf{x}_{\text{true}}$.

The comparative methods we selected include Lasso (with penalty parameter λ selected based on AIC and BIC), Logistic regression with L_1 penalty (penalty parameter chosen through cross-validation), Random Forest, Thompson Variable Selection (TVS, Y. Liu & Ročková, 2023), and FSTD (Fard et al., 2013). Since Lasso is only suitable for regression tasks, TVS is only applicable to regression and binary classification tasks, and logistic regression with L_1 penalty and FSTD are only suitable for classification tasks, we use the corresponding comparison methods based on the type of task.

ACP and TVS choose the variables whose values of θ and posterior probabilities are greater than a preset threshold relatively, which are both set to 0.9 in our simulations. Random Forest selects variables with importance score greater than the average score of all variables. FSTD chooses variables whose value functions are greater than 0. Lasso and logistic regression with L_1 penalty select variables with non-zero coefficients.

We evaluate all the compared methods using four metrics: Accuracy ($\frac{\text{TP+TN}}{\text{TP+TP+TN+FN}}$), Precision ($\frac{\text{TP}}{\text{TP+FP}}$), Recall ($\frac{\text{TP}}{\text{TP+FN}}$), and F_1 score ($\frac{\text{2precision}\cdot\text{recall}}{\text{precision+recall}}$), where TN, FN, FP and TP represent True Negative, False Negative, False Positive and True Positive respectively. These four metrics all range from 0 to 1. The larger they are, the better the performance.

Our simulation experiments are divided into three parts. The first part shows the comparison of the results of our proposed method with other methods in the cases of Table 1. The second part compares the results of ACP algorithm using a linear predictor with other methods in the case where the real function is a linear function under different settings of sample size and variable dimensionality. The third part demonstrates the impact of the model misspecification on ACP algorithm.

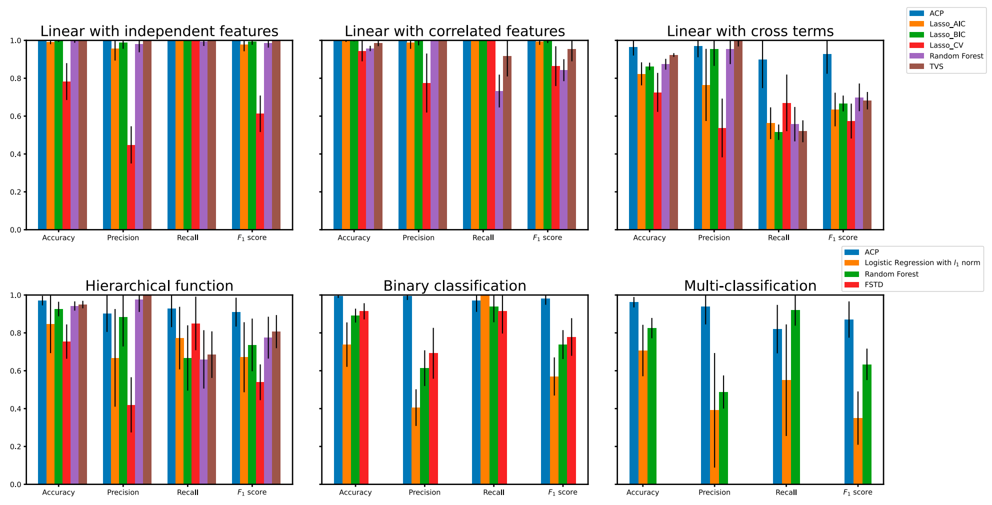

Figure 2. Results of ACP algorithm using neural-network predictor and other VS methods on different types of true functions in Table 1. $n = 200, p = 50, d = 8$. The variable selection performance is evaluated by Accuracy, Precision, Recall and F_1 score. The height of the bar is the average of 50 independent runs with the error bar on top being one standard deviation away from the mean. The legend above applies to the first four regression problems, while the legend below corresponds to the last two classification tasks.

4.1. *The ACP algorithm using neural-network predictor*

We compare the results of ACP with other VS methods for a sample size of $n = 200$ and variable dimensionality of $p = 50$. Figure 2 shows the results of 50 independent runs. In the regression problems, taking the linear function with cross terms as an example, although Lasso (BIC) and Random Forest have about the same Precision as ACP and TVS even achieves the best Precision, they have only a relatively low Recall, indicating that they miss some true variables, specifically those involved in interaction terms. ACP, on the other hand, benefits from the powerful representation capability of the neural network. Predictor can better fit the true function, and the reward can better guide the actor to select the true variables. In the classification tasks, FSTD performs very poorly in multi-class classification task, so we have not presented its results. Although ACP does not achieve the best in Recall, the Precision of other methods was very low, indicating that they selecte large numbers of irrelevant variables. Therefore, in a comprehensive manner, ACP achieves optimal performance in all six cases in terms of Accuracy and F_1 score.

4.2. *The ACP algorithm using linear predictor*

In this section we investigate the comparison among ACP using a linear predictor and other methods if the true function is indeed linear.

Figure 3 depicts the results of these methods run 50 times on linear function with correlated variables under different sample sizes. As can be seen from the figure, Lasso (CV) has a lower Precision, which indicates that it selects many irrelevant variables. Random Forest has a lower recall, showing that it ignores some true variables. Lasso (AIC) and Lasso (BIC) perform well, but are still lower in precision than ACP when the sample size is small, indicating that they still select some irrelevant variables. Additionally, TVS exhibits unstable

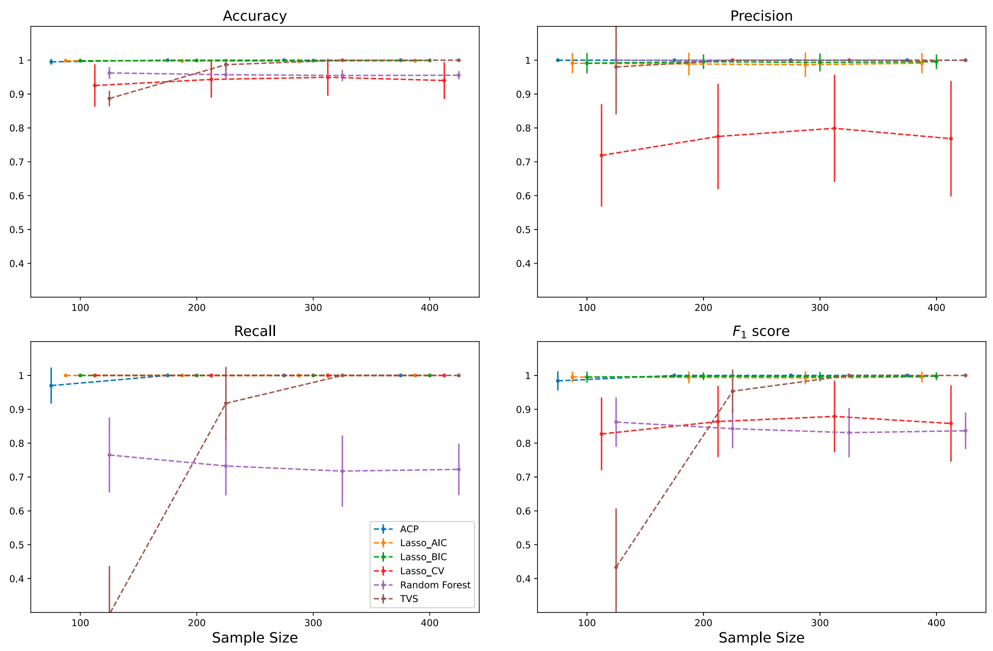

Figure 3. Results of ACP algorithm and other VS methods on linear functions with correlated variables under different sample sizes. $p = 50$, $d = 8$, and $n = \{100, 200, 300, 400\}$. The variable selection performance is evaluated by accuracy, precision, recall and F_1 score. Each error bar is drawn by 50 independent runs, with the dot being the mean and one standard deviation above and below.

Table 2. Results of ACP algorithm and other VS methods on linear functions with correlated variables under different sample sizes. $p = 300$, $d = 8$, and $n = \{100, 200\}$.

	ACP	Lasso_AIC	Lasso_BIC	Lasso_CV	Random Forest	TVS
$n = 100$						
Accuracy	1.000 ± 0.001	1.000 ± 0.000	1.000 ± 0.000	0.987 ± 0.011	0.990 ± 0.006	0.976 ± 0.003
Precision	0.982 ± 0.041	0.976 ± 0.049	0.989 ± 0.033	0.462 ± 0.201	0.488 ± 0.140	0.620 ± 0.485
Recall	0.965 ± 0.061	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.988 ± 0.037	0.100 ± 0.106
F_1 score	0.972 ± 0.035	0.987 ± 0.027	0.994 ± 0.018	0.607 ± 0.183	0.641 ± 0.129	0.167 ± 0.158
$n = 200$						
Accuracy	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.988 ± 0.012	0.998 ± 0.001	0.991 ± 0.005
Precision	0.998 ± 0.016	0.989 ± 0.033	0.998 ± 0.016	0.532 ± 0.238	0.820 ± 0.092	1.000 ± 0
Recall	0.990 ± 0.034	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.670 ± 0.204
F_1 score	0.993 ± 0.020	0.994 ± 0.018	0.999 ± 0.008	0.663 ± 0.210	0.898 ± 0.056	0.782 ± 0.161

The variable selection performance is evaluated by accuracy, precision, recall and F_1 score. The form represents mean $\pm$ std.

performance with small sample sizes, where its recall deteriorates significantly, while ACP remains excellent performance.

We also consider the case where the variable dimension is larger than the sample size. Table 2 demonstrates the results where $p = 300$ and $n = \{100, 200\}$. The conclusions are similar to the previous ones. These findings highlight the advantages of our approach, especially in scenarios with limited data availability or when identifying all relevant variables is crucial.

We also conduct experiments on linear functions with independent variables, and the results can be found in Table A2 and Figure A1 in Appendix A.1.

Table 3. Results of 50 independent runs of Lasso methods and ACP algorithm using linear and neural-network (NN) predictor on linear function with cross terms.

	ACP (linear reward)	Lasso_AIC	Lasso_BIC	Lasso_CV	ACP (NN reward)	TVS
Accuracy	0.918 ± 0.010	0.772 ± 0.094	0.898 ± 0.026	0.779 ± 0.087	0.966 ± 0.046	0.987 ± 0.001
Precision	0.969 ± 0.072	0.423 ± 0.134	0.778 ± 0.159	0.433 ± 0.149	0.971 ± 0.060	0.992 ± 0.039
Recall	0.507 ± 0.046	0.670 ± 0.124	0.550 ± 0.083	0.640 ± 0.147	0.900 ± 0.152	0.505 ± 0.024
F_1 score	0.664 ± 0.043	0.501 ± 0.100	0.635 ± 0.074	0.495 ± 0.102	0.927 ± 0.103	0.669 ± 0.023

$n = 200$, $p = 300$ and $d = 8$. The variable selection performance is evaluated by accuracy, precision, recall and F_1 score. The form represents mean $\pm$ std.

Table 4. Time consumption of a single run of ACP algorithm using neural-network (NN) predictor for different sample sizes and input dimensions on binary classification setting.

	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 1000$	$n = 2000$
$p = 50$	1 min 2 s	2 min 7 s	2 min 41 s	3 min 14 s	5 min 44 s	8 min 45 s
$p = 100$	2 min 52 s	4 min 10 s	5 min 20 s	7 min 22 s	8 min 8 s	11 min 53 s
$p = 200$	4 min 46 s	7 min 12 s	9 min 19 s	13 min 23 s	15 min 26 s	23 min 5 s
$p = 400$	6 min 44 s	16 min 10 s	20 min 41 s	25 min 40 s	28 min	45 min 9 s

4.3. *Impact of model misspecification on ACP algorithm*

Taking linear function with cross terms as an example, Table 3 compares the results of the Lasso methods, TVS and ACP algorithm using linear predictor and neural-network predictor, respectively. It can be seen that just like the Lasso methods and TVS, ACP using linear predictor does not select the variables in cross terms (Recall is low), but Precision of ACP algorithm is high, indicating that it also does not choose some irrelevant variables, while the Lasso methods select many irrelevant ones. ACP using neural-network predictor achieves simultaneously selecting true variables and screening out irrelevant ones, as previously shown in Figure 2.

4.4. *Scalability of ACP algorithm*

We have run ACP algorithm on different sample sizes and input dimensions to evaluate its scalability. The results are demonstrated in Table 4. The time complexity of ACP basically grows linearly with the input dimension and sub-linearly with the sample size. As the sample size and variable dimension increase, on one hand, ACP requires a more complex neural network predictor to obtain a better estimation of the reward function; on the other hand, the actor needs more iterations to explore different variable combinations. Therefore, for ultra-high dimensional problems, we suggest using filtering methods (such as screening) to quickly eliminate irrelevant variables before applying our method.

4.5. *Real data analysis*

We use four UCI benchmark datasets (Dua & Graff, 2017), Spambase, Communities and Crime, Madelon and CNAE-9 to validate our proposed method. The brief descriptions of theose datasets are summarized in Table 5.

Since the true covariates are usually unknown in the real world, we use the generalization error to evaluate the performance of different VS methods. For the regression and classification problems, we use the coefficient of determination (R^2) and the accuracy to measure the generalization error (calculated by 5-fold cross validation), respectively. The higher the

Table 5. UCI datasets characteristics.

Dataset	Samples	*Features*	Task
Spambase	3680	57	Binary Classification
Communities and Crime	1993	101	Regression
Madelon	2600	500	Binary Classification
CNAE-9	1080	856	Multi-Classification (9 classes)

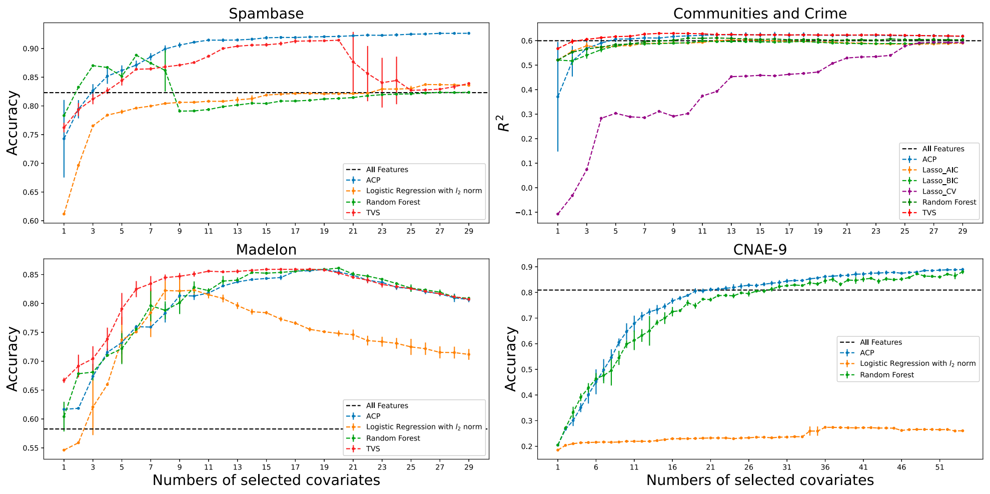

Figure 4. Performance of the different VS methods on the four real datasets in relation to the number of variables selected. Accuracy is used for the classification problem and R^2 for the regression problem. The metrics are both calculated by 5-fold cross validation. Each error bar is drawn by 5 independent runs, with the dot being the mean and one standard deviation above and below.

value, the smaller the generalization error. To be fair, all VS approaches are combined with the same end learner, a Gaussian SVM. All results are averaged over 5 independent runs.

We compare ACP with the Lasso methods, random forest, TVS, and logistic regression with L_2 norm penalty. We compare the performance of the methods when different numbers of variables are chosen. Both Lasso methods and logistic regression select variables in order of coefficients from largest to smallest, random forest uses the importance scores of the variables, TVS uses the posterior probability, and ACP is based on the value of θ for each variable.

Figure 4 illustrates the corresponding results. Firstly, it can be seen that performing variable selection does help the final prediction problem. Most of the methods achieve better results using fewer variables. Secondly, the results of the three classification problems show that the presence of redundant variables is likely to harm the performance of the classifier, so appropriate variable selection has a great impact on the final classification performance. Finally, ACP achieves the optimal performance on Spambase and CNAE-9, and performs comparably to random forest on Madelon, to TVS on Communities and Crime. In summary, ACP can achieve better results than other VS methods in real applications.

In addition, for Spambase and Madelon, we use the results of Table 2 in Fard et al. (2013) and train ACP using the same settings, and compare it with the correlation-based variable selection (CFS Hall, 2000), random forest based on Gini index (Gini-RF), and two reinforcement learning-based VS methods, FUSE (Gaudel & Sebag, 2010) and FSTD (Fard et al., 2013).

Table 6. The average performance of different VS methods runs 5 times independently on Spambase and Madelon when the number of selected variables is fixed to be 20.

	ACP	CFS	Gini-RF	FUSE	FSTD	Baseline
Spambase						
Number of selected variables	20	20	20	20	20	57
Accuracy (%)	92.10	78.53	85.48	87.11	87.19	82.93
Madelon						
Number of selected variables	20	20	20	20	20	500
Accuracy (%)	85.45	78.70	84.23	85.07	86.48	58.83

Baseline is the result of using all candidate variables. Gaussian SVM classifier is used as the end learner to obtain Accuracy.

Table 6 demonstrates the results of different VS methods on Spam and Madelon when the number of selected variables is set to 20. ACP achieves optimal performance on Spambase, and is comparable to the best FSTD on Madelon, further validating the effectiveness of the algorithm.

5. Discussion

In this work, we propose a general reinforcement learning framework to solve variable selection problems, called Actor-Critic-Predictor (ACP). We consider each variable to be chosen as a probability problem. A stochastic policy is constructed and the parameters are updated using natural policy gradient. We analyze the impact of the reward modelling error and optimization error on the final performance from both theoretical and empirical perspectives. Experiments show that ACP algorithm has a very wide range of applications and can handle both regression and classification problems, and outperforms the compared methods in most scenarios.

There are several future directions to be explored. For theoretical results, we provide an analytical framework on how different errors influence the final performance. Deeper research is worthwhile. Also, the time complexity of ACP is relatively larger than traditional methods, most of which are due to the training of neural networks. Therefore, it is worthwhile to further study how to balance the accuracy of reward and computational efficiency in practical applications, and whether a reward function can be designed so that it can satisfy both accuracy and computational efficiency.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was partially supported by the National Natural Science Foundation of China [grant numbers 12401387, 11771268, 12001356], the Fujian Alliance of Mathematics [grand number 2024SXLMQN01] and the Talent Fund Project of Fuzhou University [grand number XRC-24048].

ORCID

Yang Bai http://orcid.org/0000-0002-4660-4542

References

Agarwal, A., Kakade, S. M., Lee, J. D., & Mahajan, G. (2021). On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, *22*(98), 1–76.

Bach, F. (2008). Exploring large feature spaces with hierarchical multiple kernel learning. *Advances in Neural Information Processing Systems*, *21*, 105–112.

Barbu, A., She, Y., Ding, L., & Gramajo, G. (2016). Feature selection with annealing for computer vision and big data learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(2), 272–286. https://doi.org/10.1109/TPAMI.2016.2544315

Battiti, R. (1994). Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, *5*(4), 537–550. https://doi.org/10.1109/72.298224

Bickel, P. J., Ritov, Y., & Tsybakov, A. B. (2009). Simultaneous analysis of Lasso and Dantzig selector. *The Annals of Statistics*, *37*(4), 1705–1732. https://doi.org/10.1214/08-AOS620

Boullé, M. (2007). Compression-based averaging of selective naive Bayes classifiers. *The Journal of Machine Learning Research*, *8*, 1659–1685.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Cai, D., Zhang, C., & He, X. (2010). Unsupervised feature selection for multi-cluster data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 333–342).

Chen, J., Stern, M., Wainwright, M. J., & Jordan, M. I. (2017). Kernel feature selection via conditional covariance minimization. *Advances in Neural Information Processing Systems*, *30*, 6949–6958.

Dua, D., & Graff, C. (2017). *UCI Machine Learning Repository*. Retrieved from http://archive.ics.uci.edu/ml.

Duda, P. O., Hart, P. ., & Stork, D. G. (2000). *Pattern Classification*. Wiley Hoboken.

Fan, J., & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, *96*(456), 1348–1360. https://doi.org/10.1198/016214501753382273

Fan, J., & Lv, J. (2008). Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *70*(5), 849–911. https://doi.org/10.1111/j.1467-9868.2008.00674.x

Fang, Z., Wang, J., Geng, J., & Kan, X. (2019). Feature selection for malware detection based on reinforcement learning. *IEEE Access*, *7*, 176177–176187. https://doi.org/10.1109/Access.6287639

Fard, S. M. H., Hamzeh, A., & Hashemi, S. (2012). A game theoretic framework for feature selection. In *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery* (pp. 845–850).

Fard, S. M. H., Hamzeh, A., & Hashemi, S. (2013). Using reinforcement learning to find an optimal set of features. *Computers & Mathematics with Applications*, *66*(10), 1892–1904. https://doi.org/10.1016/j.camwa.2013.06.031

Gaudel, R., & Sebag, M. (2010). Feature selection as a one-player game. In *International Conference on Machine Learning* (pp. 359–366).

Gini, C. W. (1912). *Variability and Mutability, Contribution to the Study of Statistical Distribution and Relations*. StudiEconomico-Giuricici Della R.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*(1/3), 389–422. https://doi.org/10.1023/A:1012487302797

Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 359–366).

He, X., Cai, D., & Niyogi, P. (2005). Laplacian score for feature selection. *Advances in Neural Information Processing Systems*, *18*, 507–514.

Kakade, S. M. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems*, *14*, 1531–1538.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *The 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings*. Retrieved from http://arxiv.org/abs/1412.6980.

Kira, K., & Rendell, L. A. (1992). A practical approach to feature selection. In *Machine Learning Proceedings 1992* (pp. 249–256). Elsevier.
Lemhadri, I., Ruan, F., Abraham, L., & Tibshirani, R. (2021). LassoNet: A neural network with feature sparsity. *Journal of Machine Learning Research*, *22*(127), 1–29.
Lewis, D. D. (1992). Feature selection and feature extraction for text categorization. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23–26, 1992*.
Lin, D., & Tang, X. (2006). Conditional infomax learning: An integrated framework for feature extraction and fusion. In *European Conference on Computer Vision* (pp. 68–82).
Liu, K., Fu, Y., Wang, P., Wu, L., Bo, R., & Li, X. (2019). Automating feature subspace exploration via multi-agent reinforcement learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 207–215).
Liu, Y., & Ročková, V. (2023). Variable selection via Thompson sampling. *Journal of the American Statistical Association*, *118*(541), 287–304. https://doi.org/10.1080/01621459.2021.1928514
Mirzadeh, N., Ricci, F., & Bansal, M. (2005). Feature selection methods for conversational recommender systems. In *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service* (pp. 772–777).
Nie, F., Huang, H., Cai, X., & Ding, C. (2010). Efficient and robust feature selection via joint $l_{2,1}$-norms minimization. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2* (pp. 1813–1821).
Nie, F., Xiang, S., Jia, Y., Zhang, C., & Yan, S. (2008). Trace ratio criterion for feature selection. In *AAAI* (Vol. 2, pp. 671–676).
Peng, H., Long, F., & Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159
Shortreed, S. M., & Ertefaie, A. (2017). Outcome-adaptive Lasso: Variable selection for causal inference. *Biometrics*, *73*(4), 1111–1122. https://doi.org/10.1111/biom.12679
Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44.
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *58*(1), 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x
Varma, M., & Babu, B. R. (2009). More generality in efficient multiple kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 1065–1072).
Vidal-Naquet, M., & Ullman, S. (2003). Object recognition with informative features and linear classification. In *ICCV* (Vol. 3, p. 281).
Yang, H., & Moody, J. (1999). Data visualization and feature selection: New algorithms for nongaussian data. In *Advances in Neural Information Processing Systems* (Vol. 12).
Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks*, *94*, 103–114. https://doi.org/10.1016/j.neunet.2017.07.002
Zhang, C. (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, *38*(2), 894–942. https://doi.org/10.1214/09-AOS729
Zhang, T. (2011). Adaptive forward-backward greedy algorithm for learning sparse representations. *IEEE Transactions on Information Theory*, *57*(7), 4689–4708. https://doi.org/10.1109/TIT.2011.2146690
Zhao, Z., & Liu, H. (2007). Spectral feature selection for supervised and unsupervised learning. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 1151–1157).

Appendices

Appendix 1. Proof of Proposition 3.1

We first introduce two important lemmas, which quantify the performance difference from two different perspectives: reward modelling error and policy inconsistency.

Lemma A.1 (Performance difference due to model error): *Let* $\mathbf{r}$ *denote the true reward function and* $\hat{\mathbf{r}}$ *denote the its estimate obtained by the predictor. Let* $\rho_{\mathbf{r}}(\hat{\pi})$ *and* $\rho_{\hat{\mathbf{r}}}(\pi^*)$ *denote their objective functions respectively. Then for any given policy* π*, we have*

$$|\rho_{\hat{\mathbf{r}}}(\pi) - \rho_{\mathbf{r}}(\pi)| \leq \|\hat{\mathbf{r}} - \mathbf{r}\|_\infty.$$

Proof of Lemma A.1.:

$$\begin{aligned}|\rho_{\hat{\mathbf{r}}}(\pi) - \rho_{\mathbf{r}}(\pi)| &= \left|\mathbb{E}_{\mathbf{a}}[\hat{r}^{\mathbf{a}} - r^{\mathbf{a}}]\right| \\ &\leq \max_{\mathbf{a}} |\hat{r}^{\mathbf{a}} - r^{\mathbf{a}}| = \|\hat{\mathbf{r}} - \mathbf{r}\|_\infty.\end{aligned}$$

■

Lemma A.2 (Performance difference due to policy inconsistency): *If there are two polices* π_1 *and* π_2*, let* $\rho_{\mathbf{r}}(\pi_1)$ *and* $\rho_{\mathbf{r}}(\pi_2)$ *denote their objective functions respectively. Further assume the reward function satisfies* $||\mathbf{r}||_\infty \leq R$*. Then we have*

$$|\rho_{\mathbf{r}}(\pi_1) - \rho_{\mathbf{r}}(\pi_2)| \leq 2R \cdot D_{\mathrm{TV}}(\pi_1\|\pi_2),$$

where $D_{\mathrm{TV}}(\cdot\|\cdot)$ *is the total variation difference of two distributions.* $D_{\mathrm{TV}}(\pi_1\|\pi_2) = \sup_{\mathbf{a}} |\pi_1(\mathbf{a}) - \pi_2(\mathbf{a})|$.

Proof of Lemma A.2.: Firstly introduce a useful inequality: let $f(x)$ be a real bounded function that is $f(x) \in [-f_{\max}, f_{\max}]$, $0 \leq f_{\max} < \infty$. Let $P_1(x)$ and $P_2(x)$ denote two probability distributions over $\mathcal{X}$, and then we have

$$|\mathbb{E}_{P_1}[f(x)] - \mathbb{E}_{P_2}[f(x)]| \leq 2f_{\max} D_{\mathrm{TV}}(P_1\|P_2).$$

Next,

$$\begin{aligned}|\rho_{\mathbf{r}}(\pi_1) - \rho_{\mathbf{r}}(\pi_2)| &= \left|\sum_a \pi_1(a) r^a - \sum_a \pi_2(a) r^a\right| \\ &= \left|\mathbb{E}_{a\sim\pi_1}[r^a] - \mathbb{E}_{a\sim\pi_2}[r^a]\right| \leq 2R D_{\mathrm{TV}}(\pi_1 \mid \pi_2).\end{aligned}$$

■

Based on these two lemmas, we now can prove our main theoretical result (Proposition 3.1).

Proof of Proposition 3.1.:

$$\begin{aligned}\rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\mathbf{r}}(\pi^*) &= \rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\hat{\mathbf{r}}}(\pi^*) + \rho_{\hat{\mathbf{r}}}(\pi^*) - \rho_{\mathbf{r}}(\pi^*) \\ &\geq \rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\hat{\mathbf{r}}}(\hat{\pi}^*) + \rho_{\hat{\mathbf{r}}}(\pi^*) - \rho_{\mathbf{r}}(\pi^*) \\ &= \rho_{\mathbf{r}}(\hat{\pi}) - \rho_{\hat{\mathbf{r}}}(\hat{\pi}) + \rho_{\hat{\mathbf{r}}}(\hat{\pi}) - \rho_{\hat{\mathbf{r}}}(\hat{\pi}^*) + \rho_{\hat{\mathbf{r}}}(\pi^*) - \rho_{\mathbf{r}}(\pi^*) \\ &\geq -2R\epsilon_{\mathrm{opt}} - 2\epsilon_{\mathrm{mod}}\end{aligned}$$

The first greater than uses the definition of $\hat{\pi}^*$. ■

Appendix 2. Algorithm implementation details and more experimental results

Our implementation is mainly based on Numpy, Pytorch and scikit-learn. The Architecture of the predictor is a two-layer neural network, where the output layer is designed differently according to different problems. We use mean square error for regression problems and cross-entropy for classification problems as the generalization error. For convenience, we just use *MLPRegressor* and *MLPCLassifier* in scikit-learn, which is also faster than Pytorch implementation on CPU. We use Adam (Kingma & Ba, 2015) to optimize the actor and predictor. We have a max iteration step for training the actor, and in the meantime, we set a stopping criterion in case the algorithm converges early. When the L_2 norm of the difference of $\mathbf{w}$ in two consecutive steps is less than a threshold, we consider the algorithm

Algorithm 1 Compute the reward function $\hat{r}^{\mathbf{a}}$ corresponding to the action $\mathbf{a}$.

```
Require:
1: split the data into training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n_1}$ and validation set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n_2}$.
2: solve the optimization problem $\hat{f}_{\mathbf{a}} = \arg\min_f \frac{1}{n_1} \sum_{i=1}^{n_1} [l(y_i, f(\mathbf{x}_i^{\mathbf{a}}))]$.
3: compute the reward function $\hat{r}^{\mathbf{a}} = -\frac{1}{n_2} \sum_{j=1}^{n_2} l(y_j, \hat{f}_{\mathbf{a}}(\mathbf{x}_j^{\mathbf{a}}))$.
4: return $\mathbf{a}$, $\hat{r}^{\mathbf{a}}$ and $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a})$.
```

Algorithm 2 Actor-Critic-Predictor Algorithm.

```
Require: $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$, policy parameters $\boldsymbol{\theta}$, batch size $B$, penalty parameter for ridge regression $\gamma$, coefficient of exponentially weighted moving average $\alpha$, maximum number of iterations $M$, a small probability quantity $e = 0.02$.
1: initialize $\boldsymbol{\theta}^{(0)}$, where $\theta_j^{(0)} = 0.5, j = 1, \ldots, p$. $r_{\text{baseline}} = 0$.
2: for $m = 0, 1, \ldots, M-1$ do
3:     for $b = 1, \ldots, B$ do
4:         sample action $\mathbf{a}_b$ from policy $\pi_{\boldsymbol{\theta}}$.
5:         implement Algorithm 1 to get $\hat{r}_b$, $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_b)$.
6:     end for
7:     Based on $\{\mathbf{a}_b, \hat{r}_b, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_b)\}_{b=1}^{B}$,
8:     update the baseline: $r_{\text{baseline}} = (1-\alpha) r_{\text{baseline}} + \alpha \frac{1}{B} \sum_b \hat{r}_b$.
9:     $\mathbf{w}^{(m)} = \arg\min_{\mathbf{w}} \frac{1}{B} \sum_{b=1}^{B} [\mathbf{w}^\top \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_b) - (\hat{r}_b - r_{\text{baseline}})]^2 + \gamma \mathbf{w}^\top \mathbf{w}$.
10:    $\boldsymbol{\theta}^{(m+1)} = \boldsymbol{\theta}^{(m)} + \mathbf{w}^{(m)}$.
11:    clip $\boldsymbol{\theta}^{(m+1)}$ to $[e, 1-e]$.
12:    if convergence conditions are met, then
13:        break
14:    end if
15: end for
Ensure: $\boldsymbol{\theta}^{(m)}$.
```

Table A1. Default values of hyperparameters used in experiments.

Parameter	Value
Number of neurons for hidden layer	128
Initial learning rate for predictor	1e-2
Max iteration step for actor	500
Max training epoch for predictor	1000
Moving average weight for baseline	0.95
Batch size	64
Validation set ratio	0.3
Threshold for final output	0.9
Norm difference of **w** for convergence	1e-3
Depth of a tree for random forest	5
'gamma' for SVM with rbf kernel	'auto'

to converge and stop it. For other methods (Lasso, Logistic regression, random forest and Gaussian SVM) used in this paper, we implement them based on scikit-learn.

All hyperparameters used in experiments are summarized in Table A1.

A.1 Linear function with independent covariates

Table A2. Results of ACP algorithm and other VS methods on linear functions with independent variables under different sample sizes.

	ACP	Lasso_AIC	Lasso_BIC	Lasso_CV	Random Forest	TVS
$n = 100$						
Accuracy	0.998 ± 0.003	0.992 ± 0.007	0.996 ± 0.004	0.883 ± 0.062	0.823 ± 0.024	0.998 ± 0.002
Precision	0.945 ± 0.084	0.797 ± 0.131	0.875 ± 0.103	0.218 ± 0.085	0.130 ± 0.016	1.000 ± 0.000
Recall	0.975 ± 0.061	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.968 ± 0.060	0.917 ± 0.092
F_1 score	0.957 ± 0.058	0.881 ± 0.085	0.930 ± 0.060	0.350 ± 0.111	0.228 ± 0.025	0.954 ± 0.053
$n = 200$						
Accuracy	1.000 ± 0.001	0.999 ± 0.001	0.999 ± 0.001	0.959 ± 0.020	0.812 ± 0.020	1.000 ± 0
Precision	0.982 ± 0.041	0.867 ± 0.102	0.947 ± 0.073	0.193 ± 0.081	0.041 ± 0.004	1.000 ± 0
Recall	0.980 ± 0.046	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0
F_1 score	0.980 ± 0.032	0.926 ± 0.060	0.971 ± 0.041	0.316 ± 0.109	0.079 ± 0.007	1.000 ± 0

$p = 300$, $d = 8$, and $n = \{100, 200\}$. The variable selection performance is evaluated by Accuracy, Precision, Recall and F_1 score. The form represents mean $\pm$ std.

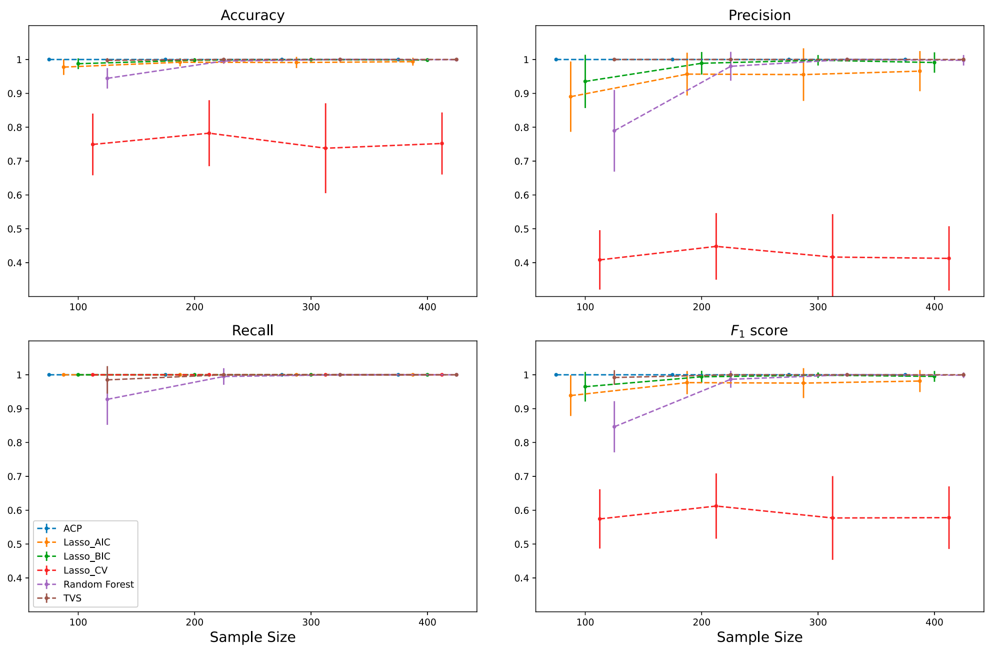

Figure A1. Results of ACP algorithm and other VS methods on linear functions with independent variables under different sample sizes. $p = 50$, $d = 8$, and $n = \{100, 200, 300, 400\}$. The variable selection performance is evaluated by Accuracy, Precision, Recall and F_1 score. Each error bar is drawn by 50 independent runs, with the dot being the mean and one standard deviation above and below.