Stochastic Local Search Heuristics for Efficient Feature Selection: An Experimental Study

Ole Jakob Mengshoel* Eirik Flogard† Jon Riege‡ Tong Yu§

Abstract

Feature engineering, including feature selection, plays a key role in data science, knowledge discovery, machine learning, and statistics. Recently, much progress has been made in increasing the accuracy of machine learning for complex problems. In part, this is due to improvements in feature engineering, for example by means of deep learning or feature selection. This progress has, to a large extent, come at the cost of dramatic and perhaps unsustainable increases in the computational resources used. Consequently, there is now a need to emphasize not only accuracy but also computational cost in research on and applications of machine learning including feature selection. With a focus on both the accuracy and computational cost of feature selection, we study stochastic local search (SLS) methods when applied to feature selection in this paper. With an eye to containing computational cost, we consider an SLS method for efficient feature selection, SLS4FS. SLS4FS is an amalgamation of several heuristics, including filter and wrapper methods, controlled by hyperparameters. While SLS4FS admits, for certain hyperparameter settings, analysis by means of homogeneous Markov chains, our focus is on experiments with several real-world datasets in this paper. Our experimental study suggests that SLS4FS is competitive with several existing methods, and is useful in settings where one wants to control the computational cost.

1 Introduction

Context. Feature selection (FS), i.e., finding the best features or attributes among a large number of them, plays an important role in machine learning (ML), knowledge discovery, and data mining [21, 10, 3, 4]. Reasons for feature selection include improved accuracy, explainability, understandability, and computational efficiency of the resulting machine learning model [21, 10]. There is an important distinction between filter and wrapper methods for feature selection [21]. The filter approach selects features in a preprocessing step, and the features selected do not depend on the ML algorithm used. The wrapper approach, in contrast, uses an ML algorithm as an integral part of the FS process. Variants of local search (such as backward selection and forward selection) have traditionally been

*NTNU, email: ole.j.mengshoel@ntnu.no
†Arbeidstilsynet, email: eirik.flogard@arbeidstilsynet.no
‡Boston Consulting Group, email: Riege.Jon@bcg.com
§CMU, email: tong.yu@sv.cmu.edu

This paper was presented at the NIK-2021 conference; see http://www.nik.no/.

employed for wrapper-based FS [10]. Methods such as genetic algorithms [18], regression [19], stochastic local search [23] and item sets [20] have also been used.

This paper encourages cross-fertilization between research on FS and stochastic local search (SLS). An SLS algorithm is a generalization of local search where stochasticity (or randomization) is also applied. Thus, SLS algorithms make occasional random search steps in order to avoid getting stuck or trapped in local but non-global optima. SLS is among the best methods to solve many computationally hard problems [15]. For example, SLS performs well in solving the satisfiability (SAT) problem [15, 35] as well as in computing the maximum a posteriori (MAP) hypothesis [28] and the most probable explanation (MPE) [22, 25] in Bayesian networks (BNs). SLS and its variants have also been widely used in other applications, such as sparse signal recovery [27], neural architecture search [34], sentence summarization [29], and subset selection [1, 30].

Problems. Clearly, recent advances in AI and ML have been impressive. At the same time, one may want to reflect on the massive computational, energy, and human resources brought to bear in today's AI and ML efforts, and how such resource consumption has recently increased. For example, OpenAI, a prominent AI development and deployment company, made the following observations on May 16, 2018:[1]

> [S]ince 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.4-month doubling time (by comparison, Moore's Law had a 2-year doubling period). Since 2012, this metric has grown by more than 300,000x (a 2-year doubling period would yield only a 7x increase).

While a 300,000x growth in compute over a 6-year period is impressive, there may also reason to be concerned. If this growth continues, where does it lead to? What are the sustainability implications? And which individuals and organizations can afford to participate in and drive AI and ML research forward, if such massive compute resources are dramatically beneficial or (even worse) required to stay competitive?

Contributions. The contributions in this paper are in part motivated by concerns about the dramatic increases in resources being consumed in AI or ML. We make resource utilization, in particular computational cost measured in terms of compute time, more of a consideration compared to much previous research. At the same time, we acknowledge that FS research has, with notable exceptions [2, 10], often been heavily experimental.

In this paper, we integrate filter and wrapper methods for FS via SLS4FS, "Stochastic Local Search (SLS) for (4) Feature Selection (FS)," and provide experimental results. Compared to the most closely related work [23], SLS4FS adds and integrates three heuristics, detailed in this paper: soft greedy search, FS filters, and a randomized neighborhood relation [26]. Key intuitions underlying the results with our hybrid SLS4FS method include (i) in FS, the computational cost of a wrapper's greedy search step is typically much greater than the computational cost of a noisy or an initialization (or filter) search step and (ii) the greedy steps are still useful in FS for refining a filter's results when optimizing a feature subset. Our experiments with SLS4FS with three different classifiers and several real-world datasets provide further details about these intuitions and demonstrate the competitiveness of SLS4FS. SLS4FS is formulated such that a Markov Chain analysis is possible (see [22, 23]), however such analysis is not pursued here.

[1] `https://openai.com/blog/ai-and-compute/`

2 The Challenges of Feature Selection

We focus here on two goals for SLS: maximizing fitness and controlling computational cost. In SLS for FS specifically, maximizing fitness corresponds to maximizing ML model accuracy, and controlling computational cost corresponds to controlling the training time of ML models. We further discuss these two FS goals below.

Maximizing Fitness. We study search spaces consisting of bit-strings $\mathbb{B} = \{0,1\}^n$. Fitness f is a pseudo-boolean function (PBF) that maps from $\mathbb{B}$ to the non-negative real numbers $\mathbb{R}_{\geq 0}$. Our focus is to optimize (without loss of generality, maximize) the fitness function f and find a global optimum $\boldsymbol{b}^*$:

$$\boldsymbol{b}^* = \arg\max_{\boldsymbol{b} \in \mathbb{B}} f(\boldsymbol{b}) . \tag{1}$$

When considering FS problems, a bit b in a bitstring $\boldsymbol{b}$ indicates whether a corresponding feature in a dataset is present ($b = 1$) or absent ($b = 0$) for purposes of learning. One can generalize (1) in order to handle multiple optima $\boldsymbol{b}_1^*$, $\boldsymbol{b}_2^*$, ... and multiple fitness (or objective) functions f_1, f_2, ... But we here keep it simple, in order not to complicate the notation and discussion.

Controlling Computational Cost. What is the time it takes for an algorithm to search for and find $\boldsymbol{b}^*$ or a good approximation? There are several factors, but let us highlight these. First, it depends on the cost g, for example compute time, it takes to compute $f(\boldsymbol{b})$ for a state $\boldsymbol{b} \in \mathbb{B}$: $g(\boldsymbol{b})$. Here, g maps from $\mathbb{B}$ to $\mathbb{R}_{\geq 0}$. Second, it depends on the complexity of the search landscape induced by f, and how that landspace interacts with the search algorithm. An SLS algorithm will in general evaluate $g(\boldsymbol{b})$ for many $\boldsymbol{b} \in \mathbb{B}$ when running. Given these factors, controlling computational cost of FS is an important goal. We are motivated by the fact that the amount of compute used in the largest AI training runs has recently increased exponentially with a 3.4-month doubling time[2] and with runtimes of days or weeks. In contrast, the compute time per experiment in Section 4 is cut off at a maximum of 100 seconds, thus enabling an exploratory and human-centric workflow that includes ML as a component.

Hyperparameters and Heuristic Settings. In order to maximize fitness and control cost as discussed above, we need to optimize SLS hyperparameter and heuristic settings. This could be done via recent hyperparameter tuning methods including Bayesian optimization (BO) [32, 31, 8]. Unfortunately, while such methods are often general and mathematically elegant, they do not always scale well [36]. We focus in this paper on fast experiments with SLS under limited computing resources, on the order of seconds or minutes, while BO typically takes hours or days.

Multi-objective optimization algorithms (MOOAs), for example multi-objective evolutionary algorithms [6, 5], may seem like another alternative to adress the above two goals related to fitness f and cost g. A MOOA would compute a Pareto front that approximates Pareto optimality, trading off the two objectives of fitness f and computational cost g of the resulting model, for example a classifier. However, we are in this paper interested in a different problem. We study the computational cost g of the SLS process itself when searching for $\boldsymbol{b}^*$, a bitstring of maximal fitness $f^* = f(\boldsymbol{b}^*)$.

Specifically, we study the effect of varying several heuristics and hyperparameters for real-world FS problems. This is a challenge, as both the FS problems and the configuration of SLS4FS are potentially high-dimensional and complex multi-modal search spaces.

[2] See above and `https://openai.com/blog/ai-and-compute/`.

3 SLS4FS: SLS for Feature Selection

We now discuss our algorithm for SLS-based feature selection, "Stochastic Local Search (SLS) for (4) Feature Selection (FS)" (or SLS4FS). The algorithm is presented briefly in previous work [26]. In this paper we provide a detailed discussion of SLS4FS, including pseudo-code, and its performance in three experiments with real-world datasets. In this section we first discuss the components or search steps of SLS4FS in Section 3.1 before presenting the algorithm's overall structure in Section 3.2.

3.1 SLS Search Steps

Local search takes place in the proximity, or in the neighborhood, of the current state $\boldsymbol{b}$, and we introduce these definitions.

Definition 1 (Neighborhood) *Let $\boldsymbol{b} = b_1 \dots b_i \dots b_n \in \{0,1\}^n$ and $\boldsymbol{b}' = b'_1 \dots b'_i \dots b'_n \in \{0,1\}^n$. Further, use $\oplus$ for exclusive or and define Hamming distance H as $H(\boldsymbol{b}', \boldsymbol{b}) = \sum_{i=1}^{n} (b'_i \oplus b_i)$. The neighborhood $N(\boldsymbol{b}) \subset \{0,1\}^n$ of $\boldsymbol{b}$ is defined as all bitstrings with a Hamming distance H of one to $\boldsymbol{b}$: $N(\boldsymbol{b}) = \{\boldsymbol{b}' \in \{0,1\}^n \mid H(\boldsymbol{b}', \boldsymbol{b}) = 1\}$.*

In FS [3] and other problems with non-trivial g, working with the neighborhood $N(\boldsymbol{b})$ may be too compute-intensive and we therefore introduce a subset of it, $N(\boldsymbol{b}, N_r)$, as follows.

Definition 2 (Randomized Neighborhood) *Let $N_r \in \mathbb{N}^+$ with $0 < N_r \leq n$. A randomized neighborhood is defined as a set $N(\boldsymbol{b}, N_r) \subseteq N(\boldsymbol{b})$:*

$$N(\boldsymbol{b}, N_r) = \{\boldsymbol{b}' \in N(\boldsymbol{b}) | \boldsymbol{b}' \text{ is picked randomly from } N(\boldsymbol{b})\}, \tag{2}$$

such that $|N(\boldsymbol{b}, N_r)| = N_r$.

"Picked randomly" in (2) means "picked uniformly at random without replacement." The computational benefit of $N(\boldsymbol{b}, N_r)$ compared to $N(\boldsymbol{b})$ is perhaps clearer when considering SLS4FS search, which we do now.

SLS4FS seeks to find or approximate an optimal state $\boldsymbol{b}^*$. This is done via repeated application of a greedy step GreedyStep, a noise step NoiseStep, and a restart step RestartStep, given the current state $\boldsymbol{b}$. These search steps are formally defined as follows.

Definition 3 (Greedy Step) *A greedy step GreedyStep($\boldsymbol{b}$, N_r, f, L, D) computes from bitstring $\boldsymbol{b}$ a bitstring $\boldsymbol{b}' \in N(\boldsymbol{b}, N_r)$ while maximizing the objective function $f(\boldsymbol{b}')$ by using the learning algorithm L with dataset D. If there is a tie in $f(\boldsymbol{b}')$ among neighbors $N(\boldsymbol{b}, N_r)$, one of these neighbor is picked uniformly at random. A* strict *greedy step GreedyStep$_s$ stays with $\boldsymbol{b}$ if $f(\boldsymbol{b}) \geq f(\boldsymbol{b}')$ for all $\boldsymbol{b}' \in N(\boldsymbol{b}, N_r)$, while a* loose *or* soft *greedy step GreedyStep$_\ell$ always moves to the best-fit neighbor.*

The GreedyStep highlights the wrapper nature [21] of SLS4FS, via its use of the learning algorithm L. Definition 3 introduces two variants of the GreedyStep. For both variants we move to a neighbor $\boldsymbol{b}' \in N(\boldsymbol{b})$. The first variant, for $N_r = n$, is a *complete* greedy step. The second variant, for $N_r < n$, is a *randomized* greedy step. This randomized variant's purpose is to reduce the computational cost of a complete greedy step. Thus, when $N_r < n$, GreedyStep($\boldsymbol{b}$, N_r, f) in Definition 3 employs randomization as follows. First, it randomly chooses N_r neighbors among $N(\boldsymbol{b})$ and then picks a neighbor maximizing the objective function $f(\boldsymbol{b}')$ among them. Intuitively, with high-dimensional

datasets (large n) and $N_r \ll n$, this provides substantial computational saving relative to using $N(\boldsymbol{b})$, at the obvious drawback of not necessarily finding the best neighbor.

The goal in FS is to find a global optimum $\boldsymbol{b}^*$ (a best feature subset) or an approximation thereof. However, in many cases there are in FS problems local but non-global optima, as demonstrated in Section 4.2, where search can get stuck. SLS4FS contains two search operators, the NoiseStep and the RestartStep, to handle this problem.

Definition 4 (Noise Step) *A noise step NoiseStep($\boldsymbol{b}$) randomly jumps from bitstring $\boldsymbol{b}$ to a neighbor $\boldsymbol{b}' \in N(b)$; note that $\boldsymbol{b}' \neq \boldsymbol{b}$.*

While there are different ways to randomize noise steps [22, 24], we focus in this paper on the simple and easy-to-analyze NoiseStep of picking a neighbor uniformly at random.

Definition 5 (Restart Step) *A restart step RestartStep(F,D) randomly computes a bitstring $\boldsymbol{b} \in \{0,1\}^n$, using a filter algorithm F with a dataset D.*

Clever restart and initialization algorithms can have a very positive impact on SLS optimization [25, 24]. For FS specifically, a filter F should ideally start search close to $\boldsymbol{b}^*$ at low computational cost. For the RestartStep, we study in this paper both naive methods and more advanced filter algorithms from the FS literature [10, 3, 4]. We consider the following three naive filter methods.

Definition 6 (Zeros Filter) *The all-zeros filter F_{0s} creates an initial feature subset in this way: We set each bit b_i to zero, $b_i = 0$, $1 \leq i \leq n$.*

Definition 7 (Ones Filter) *The all-ones filter F_{1s} creates an initial feature subset in this way: We set each bit b_i to one, $b_i = 1$, $1 \leq i \leq n$.*

Definition 8 (Uniform (at Random) Filter) *The uniform at random filter F_U creates an initial feature subset in this way: For each bit b_i, where $1 \leq i \leq n$, we flip an unbiased coin. If the coin comes up heads, we set $b_i = 1$. If it comes up tails, we set $b_i = 0$.*

The advanced filters studied can be defined as follows.

Definition 9 (Score-Based Filter) *A score-based filter creates an initial feature subset by assigning a score to every feature. For each bit b_i, if the corresponding feature's score is in the 90th percentile or above, $b_i = 1$, else $b_i = 0$.*

The following 8 score-based filters are used in this paper: mutual information (F_{MI}), ANOVA (F_{A}), χ^2 (F_{χ^2}), variance (F_{V}), random forrest impurity (F_{RFI}), random forrest permutations (F_{RFP}), lasso regression (F_{LR}) [33], and ridge regression (F_{RR}) [13]. Both the advanced, score-based filters as well as the naive F_{U}, $F_{0\mathrm{s}}$, and $F_{1\mathrm{s}}$ filters are studied experimentally in Section 4.

3.2 The SLS4FS Algorithm

The SLS4FS algorithm, summarized in Algorithm 1, searches the bitstring space $\{0,1\}^n$ representing feature subsets while optimizing accuracy f as discussed below. An optimized feature subset, represented as a bitstring $\boldsymbol{b}^+$, is the output of SLS4FS. SLS4FS is tailored to FS [26] but is based on previous SLS algorithms [22, 23].

Algorithm 1: SLS for FS (SLS4FS).

Input : Probability of restart P_r, noise step P_n, dataset D with number of instances d and features n, machine learner L, filter F, accuracy $f(\boldsymbol{b},D,L)$ for L on dataset D with subset $\boldsymbol{b}$, termination threshold τ, strict or soft GreedyStep X, number of neighbors N_r for the GreedyStep.

Output: Optimized feature subset $\boldsymbol{b}^+$

$\boldsymbol{b} \leftarrow$ RestartStep(F,D), $c \leftarrow 0$, $f^+ \leftarrow 0$, $\boldsymbol{b}^+ \leftarrow \boldsymbol{b}$
while $\neg$ *Terminate()* **do**
  $c \leftarrow c+1$
  if *Rand(0,1)* $< P_r$ **then**
    $\boldsymbol{b} \leftarrow$ RestartStep(F,D) { Def. 5 }
  else
    if *rand(0,1)* $< P_n$ **then**
      $\boldsymbol{b} \leftarrow$ NoiseStep($\boldsymbol{b}$) { Def. 4 }
    else
      $old\_b \leftarrow \boldsymbol{b}$
      $\boldsymbol{b} \leftarrow \text{GreedyStep}_X(\boldsymbol{b},N_r,f,L,D)$ { Def. 3 }
      if $old\_b = \boldsymbol{b}$ **then**
        { We may be stuck in a local optimum }
        $P_r = P_r + \bar{P}_r\alpha_r$ { Increase P_r }
        $P_n = P_n + \bar{P}_n\alpha_n$ { Increase P_n }
      else
        $P_r = P_r(1-\alpha_r/2)$
        $P_n = P_n(1-\alpha_n/2)$
  if $f(\boldsymbol{b}) > f^+$ **then**
    { Update the current best subset $\boldsymbol{b}^+$ }
    $f^+ \leftarrow f(\boldsymbol{b},D,L)$
    $\boldsymbol{b}^+ \leftarrow \boldsymbol{b}$
return $\boldsymbol{b}^+$

Input and Output. Given a machine learner L, the objective function f is computed in wrapper fashion [21]. For a feature subset $\boldsymbol{b}$, we run L on D using the features indicated by $\boldsymbol{b}$; $f(\boldsymbol{b})$ gives the estimated accuracy of L [21, 26]. The estimated accuracy is obtained by cross-validation (CV) on a training set or by validation on a separate test set.

Initialization and Search. SLS4FS starts, using RestartStep(F,D), from a random initial state $\boldsymbol{b}$ among the 2^n possible bitstrings.[3] RestartStep(F,D) initializes a feature subset using a filter method F operating on the dataset D. In each search step, SLS4FS performs (i) a GreedyStep with probability $(1-P_r)(1-P_n)$; (ii) a NoiseStep with probability $(1-P_r)P_n$; or (iii) a RestartStep with probability P_r. During search, SLS4FS keeps track of a best-so-far $\boldsymbol{b}^+$. If, for the i-th search step $f(\boldsymbol{b}) > f(\boldsymbol{b}^+)$, then $\boldsymbol{b}$ is the new best-so-far. Upon termination, SLS returns $\boldsymbol{b}^+$ as an approximation to $\boldsymbol{b}^*$. Hyperparameters α_r and α_n can be used to dynamically adapt P_r and P_r.[4]

[3] Note that (i) a deterministic initialization, for example at $\boldsymbol{b} = 0...0$ using F_{0s}, is a special case of a randomized initialization and (ii) randomized initialization may or may not be uniformly at random.

[4] To enable a homogeneous Markov chain analysis [22, 23], one can use probabilistic restart and not adapt the probability parameters P_r and P_n when running SLS4FS [23, 36].

Table 1: Datasets for experimental evaluation.

ID	Name	# Features n	# Instances d
1	breast cancer (UCI)	9	700
2	m-of-n-3-7-10 (UCI)	10	1,324
3	checklists	575	63,634
4	cleve (UCI)	9	202
5	madelon [11]	500	2,000
6	bioresponse [16]	1,776	3,751
7	gas-drift (UCI)	128	13,910
8	crime (UCI)	124	2,215

Termination. Different termination criteria can be used in SLS4FS, as suggested by Terminate() in Algorithm 1. In this work, SLS4FS terminates upon reaching an upper bound on compute time τ or when a local optimum is found.

We identify a special case of SLS4FS when only initialization is randomized:

Definition 10 (Purely greedy SLS4FS) *SLS4FS run with input parameters $P_r = 0$, $P_n = 0$, $N_r = n$, GreedyStep$_s$, $F = F_U$, and Terminate() at a local optimum is denoted* purely greedy SLS4FS.

Remark 1. A key advantage of the SLS4FS algorithm is its ability to handle local optima due to its randomized NoiseStep and RestartStep. Further, the RestartStep can provide fruitful starting points due to the use of a filter F in RestartStep(F, D). For simplicity, one can use an all-zeroes filter F_{0s}. Alternatively, a more advanced filter F may result in SLS4FS computing better feature sets.

Remark 2. When applying SLS to FS, a significant difference from many other applications of SLS is the greatly varying computational cost of search steps.

We study both of these points, and others, in experiments in Section 4.

4 Experimental Results

We conduct three experiments to analyze SLS4FS. First, in Section 4.2, we analyze several FS problems for real-world datasets to find out if they exhibit local optima. One of the advantages of SLS4FS is the possibility of it using noise and restart steps to escape local optima in the search space. In the second experiment, in Section 4.3, we evaluate the performance of SLS4FS when varying its noise parameter, restart parameter, and the filter. Third, in Section 4.4, we compare and analyze the performance of SLS4FS against other FS methods on several real-world datasets.

4.1 Datasets and Methods

Using Naive Bayes, Decision Tree, and Support Vector Machine classifiers, we conduct experiments on 8 real-world datasets to validate our hybrid SLS4FS method. For this work, we focus on small and medium sized datasets to ensure that meaningful results can be obtained within a reasonable amount of time.[5] Most of these are also real-world datasets, which are often limited in size. Table 1 lists the datasets with the number of features and instances used in our experiments. The datasets are taken from the UCI

[5]Some FS methods can take hours or even days to fully complete on one of our medium-sized datasets.

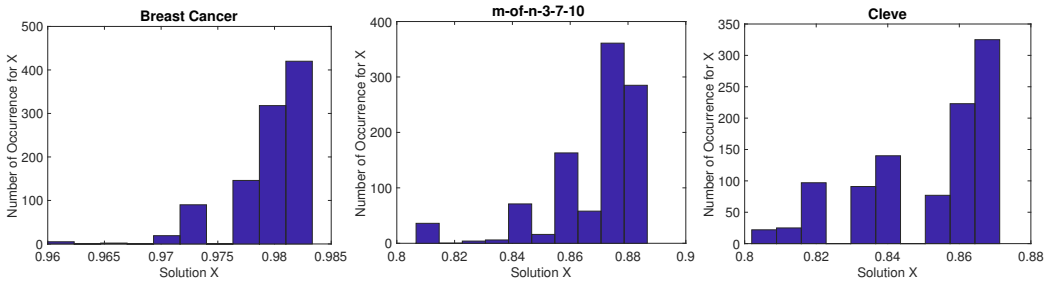

Figure 1: Experimental FS results for three datasets breast cancer, m-of-n-3-7-10, and cleve. In each panel, the x-axis shows the accuracies achieved by solutions (feature subsets) computed by SLS4FS for a dataset. The y-axis counts the number of occurrences at each accuracy level. This clearly demonstrates that there are multiple local optima in each of these FS problems.

repository[6] or the literature. We have also included a new dataset called checklists.[7] A similar dataset has also been used for constructing new checklists [9]. Experiments 2 and 3 are executed in NTNU's IDUN cluster environment, using one CPU and 24 GB of memory per run. Implementations are in Python using Scikit-learn.[8]

4.2 Experiment 1: Multiple Local Optima in Feature Selection

Goal. The challenge of local but non-global optima in FS motivated us to design the noise and restart steps in SLS4FS. To what extent does FS for real-world datasets exhibit such local optima?

Method and Data. To investigate this question, we run purely greedy SLS4FS (see Definition 10) 1,000 times for three FS problems. SLS4FS terminates upon finding a bitstring $\boldsymbol{b}^+$ with higher accuracy than all neighbors $N(\boldsymbol{b}^+)$. We use a Decision Tree model as L in SLS4FS and compute accuracy $f(\boldsymbol{b}^+)$ of the model. For each problem, we report a histogram reflecting the model's accuracy, see the plots in Figure 1.

Results and Discussion. We report the results for three small-scale problems (breast cancer, m-of-n-3-7-10, and cleve) to get a comprehensive picture. The results are shown in Figure 1. Among the 1,000 experiments, about 600 find suboptimal solutions on the breast cancer dataset. Similar observations can be made for the other two datasets; clearly m-of-n-3-7-10 is most difficult in that about 730 experiments terminate with suboptimal solutions. These results suggest that real-world FS problems contain local but non-global optima. Local optima are problematic for traditional greedy FS algorithms, like ForwardSelection and BackwardSelection. At the same time, the randomization in SLS algorithms such as SLS4FS are able to handle local optima, using $P_r > 0$ or $P_n > 0$. Motivated by these results, we carefully study the impact and optimization of P_r, P_n, and other SLS4FS heuristics in Section 4.3.

4.3 Experiment 2: Varying SLS4FS Heuristics and Settings

Goal. How do different settings of heuristics and hyperparameters impact SLS4FS's accuracy? We assess how SLS4FS performance is affected by varying these heuristics:

[6] UCI repository: `http://archive.ics.uci.edu/ml/`

[7] The checklists dataset consists of 63,634 inspections conducted by the Norwegian Labour Inspection Authority. The dataset contains 575 features related to the economical and organisational information about the target organisation. Each instance also has a binary target label, denoting whether the target organisation was found non-compliant or not at the inspection.

[8] `https://scikit-learn.org/stable/`.

Dataset	Classifier	Time τ (sec)	N_r/n
breast cancer	SVM	0.2	1.0
m-of-n-3-7-10	SVM	2.0	1.0
madelon	DT	60.0	1.0
bioresponse	NB	60.0	0.1
checklists	DT	60.0	0.1
gas-drift	DT	60.0	0.2
crime	NB	10.0	1.0

Table 2: Experimental setting in Section 4.3. The ratio of neighbors considered in the greedy step is N_r/n.

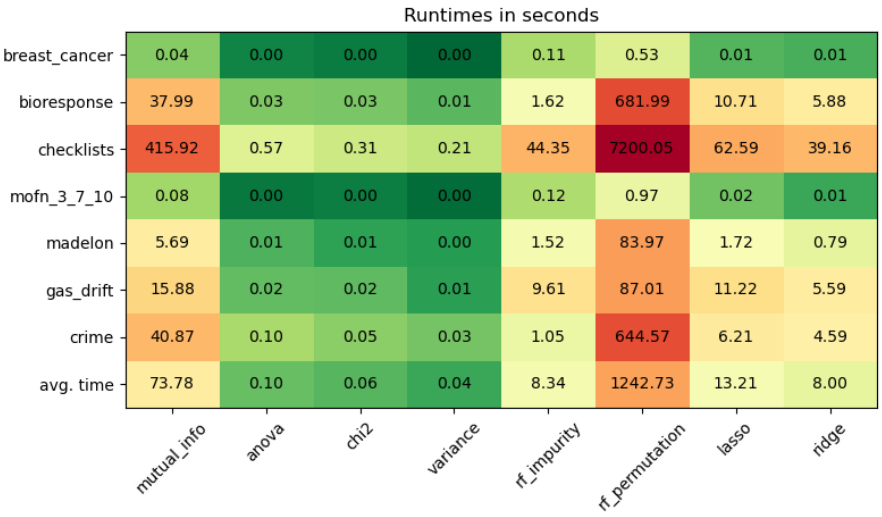

Figure 2: Runtimes for 8 different FS filters (on x-axis) on 7 different datasets (on y-axis). Runtimes are classified from extremely low (dark green) to extremely high (dark red). The filters are, left to right, mutual information (F_{MI}), ANOVA (F_{A}), χ^2 (F_{χ^2}), variance (F_{V}), random forrest impurity (F_{RFI}), random forrest permutations (F_{RFP}), lasso regression (F_{LR}) and ridge regression (F_{RR}). Per-filter runtime averages, which vary quite dramatically, are in the bottom row. Figure 3 contains corresponding accuracies.

strict versus soft Greedy, F, P_n, and P_r.

Method and Data. To inform the F-parameter of SLS4FS, we test different FS filters and record runtime and accuracy. These filters are either well-known or prominent in the FS literature [17, 4]; see Section 3. Runtimes are shown in Figure 2; F_{χ^2} or F_{V} have the best performances on almost every dataset. F_{χ^2} also has the highest recorded mean accuracy when applied to the datasets, see Figure 3. Thus, for our experiment below we use $F = F_{\chi^2}$ in our SLS4FS algorithm. Further, we include $F = F_{0\mathrm{s}}$ as a simple baseline.

Based on these filter results, we test each configuration of SLS4FS using several datasets, see Table 2. Each problem consists of a dataset, an ML method L, a compute time bound τ, and a neighborhood size N_r. The neighborhood size determines the fraction of neighbors included in the randomized neighborhood. The time limit and neighborhood size are set to limit the overall running time of SLS4FS. SLS4FS hyperparameters and heuristics that are varied are: P_n, P_r, F, and $\mathsf{GreedyStep}_s$ versus $\mathsf{GreedyStep}_\ell$.

Every evaluation in the experiment is done by restricting the dataset based on the SLS4FS-selected features, initializing the ML method, training the ML method on 2/3 of the dataset and calculating the accuracy based on the other 1/3.

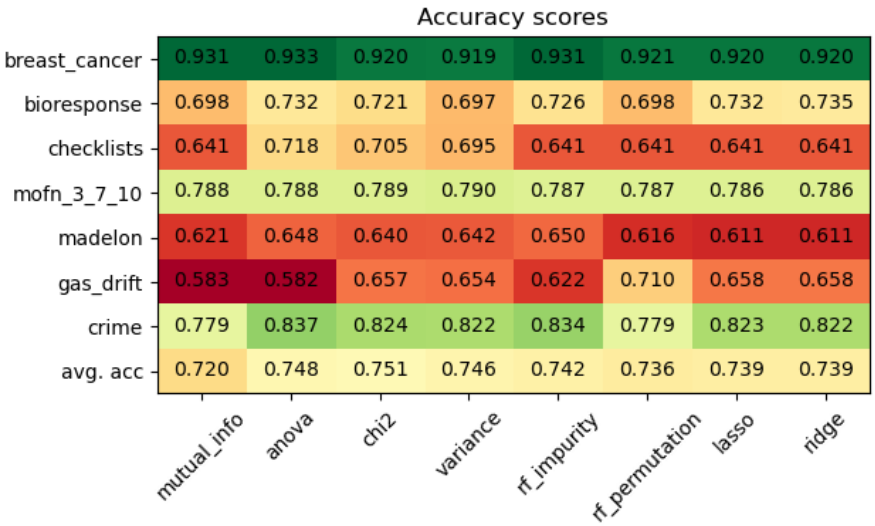

Figure 3: Accuracy scores for 8 different FS filters (on x-axis) on 7 different datasets (on y-axis). Per-filter accuracy averages are in the bottom row. While χ^2 (F_{χ^2}) has the highest average accuracy, the differences in averages for the best methods are very small and not statistical significant. This table corresponds to the table with runtimes in Figure 2.

Results and Discussion. Four configurations of SLS4FS, each tested with 20 different values of noise probability P_n, are shown in Figure 4.[9] The plots are consistent with findings about local optima in Section 4.2 as well as previous Markov chain analyses and experiments with SLS [14, 22, 25]: P_n being varied clearly has a significant impact on performance. And the optimal noise probability varies between different problem instances. Plots 4(a) and 4(b) do not show large differences between strict GreedyStep$_s$ and soft GreedyStep$_\ell$. However, soft outperforms strict on the m-of-n-3-7-10 dataset, and in general seems to be at least as good. As seen in Plot 4(d), restart has little effect on most problems, and a slight negative effect on a few. This is a small surprise, given previous research highlighting the benefit of restart [22, 25], and an area for future research.

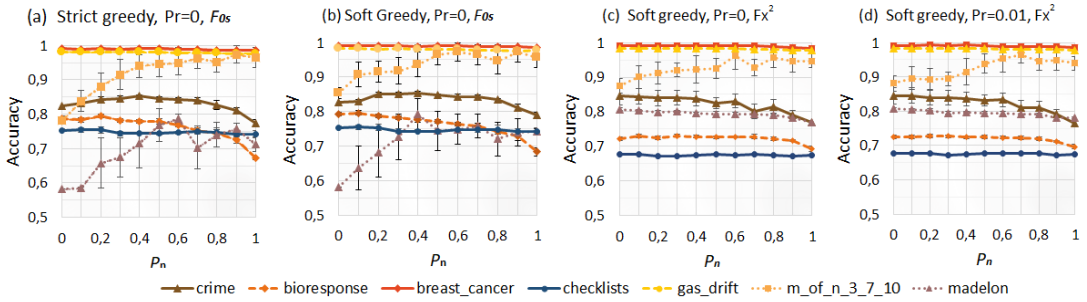

Figure 4: Accuracy (on y-axis) versus P_n (on x-axis) for four configurations of SLS4FEVE (in plots (a), (b), (c), and (d)) as applied to six datasets. Each datapoint in a plot is the average over 10 runs, with the error bars showing the 95% confidence interval.

Subjectively, we deem these to be the most promising configurations of SLS4FS: soft Greedy, $P_r = 0$, $P_n \in [0.1, 0.5]$, and with either $F = F_{0s}$ or $F = F_{\chi^2}$ as filter depending on the selected dataset. These results inform our experiments with four different SLS4FS configurations in Section 4.4.

[9] Other configurations were also tested, but the ones in Figure 4 illustrate the main findings well.

Algorithm	Configuration
RFE	N/A
ForwardSelection (F_{0s})	N/A
BackwardSelection (F_{1s})	N/A
AdaptiveNoise	$\phi = 0.2$, $\theta = 1/6$
AdaptiveSLS	$P_n = 0.0$, $P_r = 0.0$, $\alpha_n, \alpha_r = 0.32$
SoftSLS	$P_n = 0.5$, $P_r = 1/n$, $\alpha_n, \alpha_r = 0.0$
SLS4FS (F_{χ^2})	$P_n = 0.5$, $P_r = 0.0$, $\alpha_n, \alpha_r = 0.0$, F_{χ^2}
SLS4FS ($P_r = 0.1$)	$P_n = 0.5$, $P_r = 0.1$, $\alpha_n, \alpha_r = 0.0$, F_{χ^2}
SLS4FS (F_{0s})	$P_n = 0.5$, $P_r = 0.0$, $\alpha_n, \alpha_r = 0.0$, F_{0s}
SLS4FS (adaptive)	$P_n = 0.0$, $P_r = 0.0$, $\alpha_n, \alpha_r = 0.32$, F_{χ^2}

Table 3: Algorithms and configurations used in Section 4.4. RFE [12], AdaptiveNoise [14], AdaptiveSLS [23], and SoftSLS [23] are from the literature, the other algorithms are well-known or described in this paper.

4.4 Experiment 3: Comparing SLS4FS to Other Algorithms

Goal. How does SLS4FS compare to other FS wrappers using local search?

Method and Data. We test four configurations of SLS4FS and six other algorithms, recording the accuracy of the best feature subset for each algorithm (see Table 3). The algorithms are tested on problem instances from Table 1 and evaluations are performed similar to in Section 4.3. However, a time limit of τ = 100 sec is used in every problem instance, Here, $N_r = \lceil n/10 \rceil$ and $\mathsf{GreedyStep}_\ell$ are used for SLS4FS while $\alpha_n = \alpha_r = 0.32$ are used for AdaptiveSLS and SLS4FS (adaptive).[10]

Results and Discussion. The results are summarised in Table 4.[11] The top three performers, ranked by mean accuracy (in the right-most column), are all variations of SLS4FS. SLS4FS is quite robust across all problems compared to existing algorithms. For example, ForwardSelection achieves the highest accuracy for three problems but is far behind for other problems, leading it to be ranked as one of the last overall.

For the checklists dataset, there were differences in the number of features that were found by the best performing FS configurations. On average, ForwardSelection selected 8 features while AdaptiveSLS and SoftSLS selected 7 features. SLS4FS (F_{0s}) selected 47 features with similar performance in terms of accuracy. The same pattern was also observed for the other large datasets when comparing SLS4FS to ForwardSelection and the existing SLS algorithms. Generally, by using a randomized neighborhood, SLS4FS is able to take more steps per time unit and explore a larger fraction of the search space which yields more features with approximatly the same computational cost.

5 Conclusion and Future Work

In this paper, we adapt and apply stochastic local search (SLS) to the problem of feature selection. We study an SLS algorithm SLS4FS for feature selection; it is a hybrid approach that integrates the well-known filter and wrapper approaches. Relative to the most closely related research [23], SLS4FS adds and integrates three heuristics: soft greedy search, filters, and a randomized neighborhood relation [26]. Experimentally, motivated by constraining computational resources, we study different FS filter algorithms with SLS4FS and three ML classifiers: Decision Tree, Naive Bayes,

[10] The hyperparameters α_n and α_r were optimized empirically in pilot studies. In the pilot studies both synthetic and real-world problems were used. In the results reported here, α_n and α_r are kept constant.

[11] The main conclusion, but not the dataset-specific results, of this table has been presented earlier [26].

Table 4: Mean accuracy and standard deviation for 4 versions of SLS4FS and 6 other algorithms, applied to 7 datasets. The two right-most columns show Mean accuracy (higher is better) and Rank (lower is better) for all 10 algorithms across the 7 datasets. Overall, SLS4FS (F_{0s}) is the best algorithm among the 10 for these datasets.

	breast cancer	**m-of-n-3-7-10**	**madelon**	**biores-ponse**	**check-lists**	**gas-drift**	**crime**	**Mean**	**Rank**
RFE	0.978 ± 0.000	**1.000 ± 0.000**	0.738 ± 0.000	0.636 ± 0.000	NA	0.976 ± 0.000	0.808 ± 0.000	0.734	9
ForwardSelection	0.991 ± 0.000	0.780 ± 0.000	0.582 ± 0.000	**0.797 ± 0.000**	**0.751 ± 0.000**	**0.982 ± 0.000**	0.818 ± 0.000	0.803	8
BackwardSelection	**0.996 ± 0.000**	**1.000 ± 0.000**	0.726 ± 0.000	0.628 ± 0.000	0.675 ± 0.000	0.970 ± 0.000	0.784 ± 0.000	0.826	6
AdaptiveNoise	**0.996 ± 0.000**	**1.000 ± 0.000**	0.731 ± 0.029	0.620 ± 0.017	0.667 ± 0.000	0.973 ± 0.002	**0.854 ± 0.004**	0.834	5
AdaptiveSLS	**0.996 ± 0.000**	**1.000 ± 0.000**	0.732 ± 0.024	0.623 ± 0.025	**0.751 ± 0.000**	0.974 ± 0.003	0.830 ± 0.007	0.844	4
SoftSLS	**0.996 ± 0.000**	**1.000 ± 0.000**	0.796 ± 0.005	0.626 ± 0.023	0.749 ± 0.001	0.974 ± 0.003	0.848 ± 0.007	0.856	3
SLS4FS (F_{χ^2})	**0.996 ± 0.000**	**1.000 ± 0.000**	0.803 ± 0.006	0.734 ± 0.008	0.678 ± 0.002	**0.982 ± 0.002**	0.845 ± 0.002	0.862	2
SLS4FS ($P_r = 0.1$)	**0.996 ± 0.000**	**1.000 ± 0.000**	0.795 ± 0.006	0.717 ± 0.008	0.673 ± 0.000	**0.982 ± 0.001**	0.829 ± 0.003	0.856	3
SLS4FS (F_{0s})	**0.996 ± 0.000**	**1.000 ± 0.000**	**0.804 ± 0.012**	0.774 ± 0.005	0.749 ± 0.003	0.980 ± 0.002	0.843 ± 0.004	**0.878**	**1**
SLS4FS (adaptive)	**0.996 ± 0.000**	0.790 ± 0.019	0.796 ± 0.008	0.724 ± 0.007	0.677 ± 0.001	**0.982 ± 0.001**	0.684 ± 0.009	0.807	7

and Support Vector Machine. SLS4FS produces competitive results on several different datasets in experiments, at modest computational cost, thus reflecting our goals of maximizing model accuracy while controlling training time.

These are a few areas for future work: First, it would be useful to more systematically vary SLS4FS hyperparameters including P_r and P_n. Second, we plan to investigate other classifiers and even larger datasets with more features. Third, it would be interesting to automatically optimize the hyperparameters of SLS4FS [7, 23, 36], considering the varying computational costs of search steps for different computers and datasets. Fourth, it could be interesting to compare SLS4FS to other methods, such as tabu search, simulated annealing, and evolutionary algorithms. Fifth, we plan to further study SLS4FS from a theoretical perspective, using Markov chain hitting time analysis.

References

[1] C. Bian, C. Feng, C. Qian, and Y. Yu. An efficient evolutionary algorithm for subset selection with general cost constraints. In *Proc. AAAI*, pages 3267–3274, 2020.

[2] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. A review of feature selection methods on synthetic data. *Knowledge and Information Systems*, 34(3):483–519, 2013.

[3] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. *Feature Selection for High-Dimensional Data*. Springer, 2015.

[4] A. Bommert, X. Sun, B. Bischl, J. Rahnenführer, and M. Lang. Benchmark for filter methods for feature selection in high-dimensional classification data. *Computational Statistics & Data Analysis*, 143:1–19, 2019.

[5] C. A. C. Coello, G. B. Lamont, and D. A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag, 2006.

[6] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., 2001.

[7] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Designing fast absorbing Markov chains. In *Proc. AAAI*, pages 849–855, 2014.

[8] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proc. ICML*, pages 1436–1445, July 2018.

[9] E. L. Flogard, O. J. Mengshoel, and K. Bach. Bayesian feature construction for case-based reasoning: Generating good checklists. In *Proc. ICCBR*, pages 94–109. Springer, 2021.

[10] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, 2003.

[11] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Proc. NIPS*, pages 545–552, 2004.

[12] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[13] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[14] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. AAAI*, pages 655–660, 2002.

[15] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, 2005.

[16] B. Ingelheim. Predicting a biological response. Mar 2012.

[17] A. Jović, K. Brkić, and N. Bogunović. A review of feature selection methods with applications. In *Proc. MIPRO*, pages 1200–1205, 2015.

[18] M. M. Kabir, M. Shahjahan, and K. Murase. A new local search based hybrid genetic algorithm for feature selection. *Neurocomputing*, 74(17):2914–2928, 2011.

[19] S. Y. Kim and E. Xing. Feature selection via block-regularized regression. In *Proc. UAI*, pages 325–332, 2008.

[20] A. J. Knobbe and E. K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *Proc. KDD*, pages 237–244, 2006.

[21] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[22] O. J. Mengshoel. Understanding the role of noise in stochastic local search: Analysis and experiments. *Artificial Intelligence*, 172(8-9):955–990, 2008.

[23] O. J. Mengshoel, Y. Ahres, and T. Yu. Markov chain analysis of noise and restart in stochastic local search. In *Proc. IJCAI*, pages 639–646, 2016.

[24] O. J. Mengshoel, D. Roth, and D. C. Wilkins. Portfolios in stochastic local search: Efficiently computing most probable explanations in Bayesian networks. *Journal of Automated Reasoning*, 46(2):103–160, 2011.

[25] O. J. Mengshoel, D. C. Wilkins, and D. Roth. Initialization and restart in stochastic local search: Computing a most probable explanation in Bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):235–247, 2011.

[26] O. J. Mengshoel, T. Yu, J. Riege, and E. Flogard. Stochastic local search for efficient hybrid feature selection. In *Proc. GECCO*, pages 133–134, 2021.

[27] D. K. Pal and O. J. Mengshoel. Stochastic CoSaMP: Randomizing greedy pursuit for sparse signal recovery. In *ECML-PKDD*, pages 761–776, 2016.

[28] J. D. Park and A. Darwiche. Complexity results and approximation strategies for MAP explanations. *JAIR*, 21:101–133, 2004.

[29] R. Schumann, L. Mou, Y. Lu, O. Vechtomova, and K. Markert. Discrete optimization for unsupervised sentence summarization with word-level extraction. In *Proc. ACL*, pages 5032–5042, 2020.

[30] K. Shang, H. Ishibuchi, and W. Chen. Greedy approximated hypervolume subset selection for many-objective optimization. In *Proc. GECCO*, pages 448–456, 2021.

[31] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. NIPS*, pages 2951–2959, 2012.

[32] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. ICML*, pages 1015—-1022, 2010.

[33] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[34] C. White, S. Nolen, and Y. Savani. Exploring the loss landscape in neural architecture search. In *Proc. UAI*. AUAI Press, 2021.

[35] E. Yolcu and B. Póczos. Learning local search heuristics for boolean satisfiability. In *Proc. NeurIPS*, pages 7990–8001, 2019.

[36] T. Yu, B. Kveton, and O. J. Mengshoel. Thompson sampling for optimizing stochastic local search. In *Proc. ECML-PKDD*, pages 493–510, 2017.