Enforcing Constraints for Machine Learning Systems via Declarative Feature Selection: An Experimental Study

Felix Neutatz
f.neutatz@tu-berlin.de
TU Berlin

Felix Biessmann
fbiessmann@beuth-hochschule.de
Einstein Center Digital Future Berlin
Beuth University Berlin

Ziawasch Abedjan
abedjan@dbs.uni-hannover.de
Leibniz Universität Hannover
L3S Research Center

ABSTRACT

Responsible usage of Machine Learning (ML) systems in practice does not only require enforcing high prediction quality, but also accounting for other constraints, such as fairness, privacy, or execution time. One way to address multiple user-specified constraints on ML systems is feature selection. Yet, optimizing feature selection strategies for multiple metrics is difficult to implement and has been underrepresented in previous experimental studies. Here, we propose *Declarative Feature Selection* (DFS) to simplify the design and validation of ML systems satisfying diverse user-specified constraints. We benchmark and evaluate a representative series of feature selection algorithms. From our extensive experimental results, we derive concrete suggestions on when to use which strategy and show that a meta-learning-driven optimizer can accurately predict the right strategy for an ML task at hand. These results demonstrate that feature selection can help to build ML systems that meet combinations of user-specified constraints, independent of the ML methods used.

1 INTRODUCTION

Many modern software systems rely on Machine Learning (ML) components for automated decision making [9, 10, 16, 23, 31, 33, 60, 61, 69, 72]. Especially when used in production software systems, maintaining and monitoring ML components is important to ensure reliable predictions. A central challenge in this context is the validation of certain data properties, especially those that are due to data transformations induced by an ML model. Examples of such properties are the differential privacy or fairness of ML model predictions. Violating these constraints in production systems can have devastating consequences. But automatically enforcing these constraints is challenging. Many approaches in the ML literature focus on adapting specific models to specific constraints. For instance, Donini et al. propose a general framework for empirical risk minimization under fairness constraints [18]. And McSherry et al. introduce differential privacy for a recommender system [39]. Both examples enforce privacy and fairness by building dedicated models. Yet, when building a production system, it is often difficult to enforce these or other constraints by simply exchanging a model. One reason is that adding a new constraint or combining multiple constraints would require developing a new model. Another line of work used preprocessing and model hyperparameter optimization (HPO) to satisfy fairness constraints [52, 57]. Alternatively, such constraints can be enforced by restricting the features used by the ML component. Violations of fairness or privacy constraints are often due to just a small fraction of features. Determining them independently of an ML task is not always possible. But when removing them from the training data set, any downstream ML model can be forced to meet a given level of privacy or fairness. Hence, feature selection (FS) can be considered an attractive model-agnostic alternative to model-specific solutions for each individual constraint. Inspired by Perrone et al. [52], Figure 1 illustrates that, across models, FS affects the trade-off of accuracy with the fairness metric equal opportunity (EO) [29], feature set size, and safety levels. Each dot in this figure corresponds to a feature subset of the COMPAS dataset [35]. The charts show that independent of the model, one can achieve very different trade-offs of accuracy with metrics, such as EO [29] and safety against adversarial examples [15, 47], by just changing the original feature set.

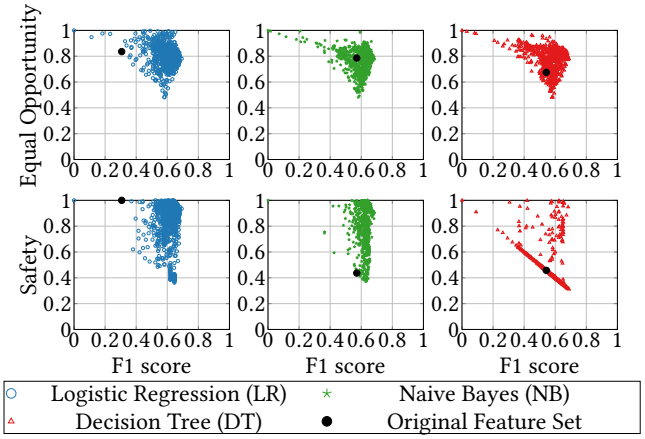

Figure 1: Accuracy trade-off with three nonfunctional metrics for LR, NB, and DT on the COMPAS dataset. Each dot corresponds to results with a different feature subset.

FS [17, 37, 41, 71] has been widely studied to address the curse of dimensionality, avoid overfitting, and accelerate training/prediction while reducing the memory footprint [27]. Therefore, 78% of all 1,233,841 Scikit-learn [51] pipeline runs in the open science platform OpenML [67] employ FS. Yet, the way FS is built into these ML pipelines renders it difficult to leverage FS for enforcing the above constraints automatically. Most ML libraries and database research on feature engineering [2–4, 30, 34, 44, 74] only support optimization of a single objective, i.e., prediction performance in terms of classification or regression losses.

In this study, we argue that decoupling the constraint validation from the ML component has decisive advantages for automated training, validation, and monitoring of software systems featuring ML components. We propose *Declarative Feature Selection* (DFS) as a model-agnostic approach to validation of ML components for user-specified constraints. One main advantage of DFS is that it is more convenient to use than building a novel ML model for

each constraint, as proposed by Donini et al. [18, 39]. Lowering the adoption threshold of systems that allow for more responsible usage of ML systems is a crucial requirement for addressing the ethical challenges that we are facing as a scientific community and as a society in general, as highlighted by Bender et al. [6]. Further, we also argue that the modularity of the proposed system, meaning enforcing constraints on ML systems by focusing on the features ingested by an ML system, has decisive advantages for the technical debt and maintenance cost of ML systems: Our approach allows to conveniently exchange the ML model component while being compliant with the constraints enforced via FS. We demonstrate the efficacy of this approach and the transferability of constraints across different model classes in a comprehensive suite of experiments.

Considering the large number of existing FS strategies, choosing the fastest and most useful strategy for the aforementioned optimization goals is not trivial. For instance, it is not clear whether FS strategies that optimize accuracy, such as the χ^2-based ranking [38], are suited to find feature subsets that satisfy constraints such as safety against adversarial attacks and fairness. Another open question is whether strategies that consider a smaller search space, such as ranking-based approaches, are suitable for as many ML scenarios as strategies that consider a larger search space of feature subsets, such as sequential selection approaches [1, 28]. To develop a generic DFS approach applicable to a diverse set of constraints, we conducted a systematic evaluation of FS strategies. To this end, we had to overcome three general challenges:

(1) **Diverse FS metrics.** To the best of our knowledge, no FS strategy was evaluated across a diverse metric set that goes beyond accuracy and runtime. Especially, competing metrics, e.g., greater privacy often leads to lower accuracy, aggravate this challenge.
(2) **Choice of constraint sets.** Each constraint refers to a threshold for a specific metric, such as EO > 90%. For each FS task, there is a set of constraints, each with an infinite space for possible thresholds. In the absence of established benchmarks, which exist for database research in the framework of TPC [64], we had to model our experiments in a way that captures a wide range of constraint sets on a diverse set of datasets.
(3) **Algorithm adaptation.** For each selected algorithm, we had to adapt it to the use case of Dfs.

Contributions. We conduct a comprehensive evaluation of 16 FS strategies on 19 binary classification tasks each with a single binary protected attribute from the OpenML platform [67]. Our results indicate which strategy is best suited for which ML scenario. We refer to the fraction of ML scenarios where an FS strategy found a feature set that satisfies all constraints as the *coverage*. Furthermore, we explore under which circumstances a strategy yields the solution the fastest. Finally, we investigate to what extent it is possible to learn the applicability of an FS algorithm. To this end, make the following contributions:

- We classify 16 FS strategies in the context of Dfs and provide a taxonomy for ML application constraints.
- We run extensive experiments with the strategies on different datasets and evaluate both search time and coverage. We explore 3318 combinations of 6 ML application constraints, 3 classification models, and 19 datasets.
- From our experimental results, we derive concrete suggestions on when to use which strategy and test a meta-learning-driven Dfs optimizer that automatically makes this decision depending on the user-specified constraints and the dataset at hand. We also provide all implementations, datasets, and the evaluation framework in our repository [45].

Main Findings. Our study lets us draw the following conclusions:

- Dfs is a promising framework to enforce a wide range of ML application constraints transparently for the user and avoids heavy ML engineering for constraint-specific models. Further, the user can add, remove, or modify constraints declaratively without engineering. As FS is orthogonal to modeling, one can also combine Dfs and constraint-specific models if required. We also show that, in many cases, the constraints still hold after switching to another ML model.
- Sequential forward selection is well-suited for the scenario of Dfs because most constraints limit the size of satisfying feature sets and these strategies can quickly find features for small sets.
- Ranking-based FS strategies that were mostly designed for accuracy perform well if the metric favors reduction and compression of the feature vector. This is the case for metrics, such as complexity, privacy, and safety. However, if the metric requires the selection strategy to prune specific features that are unrelated to accuracy, such as biased features in the case of fairness, accuracy-optimized rankings fail for high thresholds and we need to resort to strategies that consider a larger search space, such as forward selection or multi-objective NSGA-II [71].
- Our benchmark shows that the performance of FS strategies is highly dependent on the constraints, the dataset, and the classification model. This finding motivated us to leverage meta-learning to predict the most promising FS strategy for a given ML scenario. This Dfs optimizer has a 10% higher coverage on average than the best individual FS approach and achieves consistently good coverage leading to 4% less standard deviation.
- If the user has more computing resources, one can improve the coverage by running multiple strategies in parallel. By running only 5 strategies in parallel, we achieve already 94% coverage or receive in 52% of the cases the fastest result depending on whether we optimize for coverage or speed.

2 DECLARATIVE FEATURE SELECTION

We first formally define the problem of Dfs. Then, we describe its general workflow.

2.1 Problem Statement

We formulate the problem of Dfs as follows: A user specifies the ML scenario Z as the tuple $(\phi, D, D_{\text{train}}, D_{\text{val}}, D_{\text{test}}, C)$ that comprises the ML classification model ϕ, the dataset D, the strategy how to split the data into a training set D_{train}, validation set D_{val}, and test set D_{test}, and the set of ML application constraints $C = \{c_1, ..., c_M\}$. The dataset D comprises the feature set $F = \{f_1, ..., f_N\}$ and the prediction target t. Here, we focus on binary classification. A feature subset F' satisfies an ML scenario Z if no user-specified constraint $c_m \in C$ is violated. The goal of an FS strategy s is to find one feature

subset $F' \subseteq F$ that satisfies the user-specified ML scenario Z:

$$s(Z) = \begin{cases} F', & \text{if a } Z\text{-satisfiable } F' \text{ exists} \\ \emptyset, & \text{otherwise.} \end{cases}$$

In this paper, we limit our problem to finding a feature set that satisfies all user-specified constraints instead of finding all Pareto-optimal feature sets because the latter requires significantly higher computation time. Alternatively, in addition to satisfying all specified constraints, we can extend the problem statement by allowing for maximizing specified utilities subject to specified constraints. We evaluate both cases in our experiments.

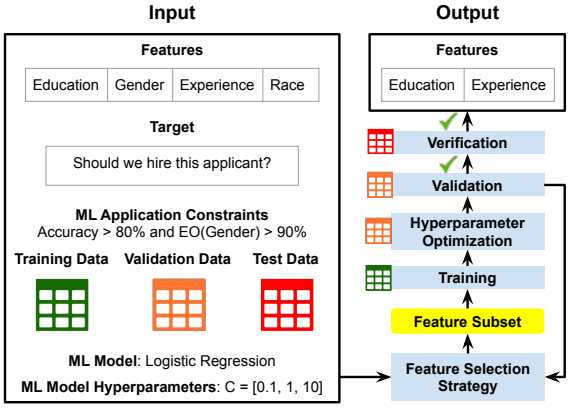

Figure 2: DFS workflow.

2.2 Workflow

Figure 2 illustrates the workflow of a typical DFS scenario. The data scientist first specifies the ML task. This task includes the dataset that contains the features and the classification target, and how to split the data into training, validation, and test. Then, the ML model is specified - here, LR. Finally, the scientist defines the ML application constraints, which are described in Section 3.

Based on the user input, an FS strategy proposes a feature subset. In Section 5, we explore a meta-learning-driven optimizer that chooses the strategy that is most likely to satisfy the ML scenario based on the experience of previous runs. Note that the optimizer only chooses an FS strategy and does not automatically generate a feature ranking. Then, the model is trained using the proposed feature subset. If the user specified a model hyperparameter space, optimization will be applied at this stage and returns the model that yielded the best accuracy score for validation. If all constraints are satisfied on the validation data, the feature subset is also evaluated on the test data. If any constraint was not satisfied, the FS strategy uses the validation loss to search for the next feature subset, as described in Section 4.3. If all constraints are also satisfied on the test data, the DFS system was successful and returns the feature subset to the user. Otherwise, DFS was not successful.

3 ML APPLICATION CONSTRAINTS

There are many metrics proposed in the ML literature besides accuracy, such as fairness [29], privacy [20], and robustness [26]. A metric translates into a constraint as soon as a threshold is specified. To choose the right set of metrics, we tried to identify major properties according to which we can distinguish constraint types and select representative ones. A very recent survey [75] counts the same set of metrics as properties that most ML applications are tested on. While in this study, we limit the prototype to a selected set of constraints, our framework is applicable to any metric that produces a numeric score based on a dataset and an ML model.

Table 1: Constraint taxonomy.

Constraint	Evaluation Dependence	#Feature Dependence	Input: Features	Input: Target	Input: Model	Input: Predictions
Max Search Time		None				
Max Feature Set Size		- -	✓			
Max Training Time	✓	- -				
Max Inference Time	✓	- -				
Min Accuracy	✓	+		✓		✓
Min Equal Opportunity	✓	-	✓	✓		✓
Min Privacy		- -				
Min Safety	✓	- -	✓	✓	✓	✓

In the taxonomy presented in Table 1, we identify three very general characteristics that can serve in distinguishing ML constraints for FS: the evaluation dependency, the feature set size dependency, and the required input. Constraints can be evaluation dependent and evaluation independent. For example, one can only verify whether an accuracy constraint is satisfied after training a model and applying this model on the test set. Therefore, accuracy is dependent on the evaluation. Accordingly, fairness and safety against adversarial examples are evaluation dependent. On the other hand, feature complexity, e.g. the number of features, differential privacy, and the user-specified budgeted time for search are independent of the evaluation. For example, if the user specifies a maximum number of 5 features, one can immediately prune all feature subsets with more than 5 features without evaluation.

Also, the correlation to the number of features differs across constraints. Accuracy on average benefits from a higher number of features. EO might negatively correlate because sensitive or biased features have to be removed. For safety against adversarial examples, our empirical study shows a larger negative correlation because the more features we use, the more options a potential adversary has to fiddle with the feature vector. Finally, the computation of different constraints requires different inputs. For instance, to identify whether a constraint that is based on feature complexity is satisfied, we only have to count the number of features that are present in the feature subset. Therefore, the features are the only required input. To calculate accuracy-related constraints, one only requires the target values and the model predictions. For fairness-based constraints, we need the features in addition to all inputs that were required for accuracy because we need to know which instances belong to the minority group and which ones to the majority group. For constraints that address the safety against adversarial examples, we additionally need the corresponding trained classification model because the robustness measures will query the trained model for instances with modified feature values.

We selected eight metrics that cover different aspects of the properties in Table 1. In the following, we describe how we measure and implement each of them as constraints in our experiments.

Min Accuracy. The model accuracy denotes how accurately the model can predict for a certain task. In this paper, we focus on binary classification tasks. We use the F1 score on binary classification tasks because it is robust against class imbalance.

Max Search Time specifies the maximum allowed time for FS.

Max Feature Set Size. One way to measure the complexity of a feature set is to count the number of its features. The number of features could also be used as a proxy metric to measure interpretability [53]. For example, a DT with 5 features is easier to understand than one that uses 100 features.

Max Training / Inference Time. For fast iterative ML development, the user might prefer models that train faster or, for real-time ML applications, the user might prefer models that predict faster. Given that the remaining ML pipeline does not change, for FS, the training and inference time directly depends on the number of features. We can easily learn this relationship and map this evaluation-dependent constraint to an evaluation-independent constraint. This optimization reduces the search space and thereby can accelerate the search. Since both Max Training / Inference Time and Max Feature Set Size constraints are two approaches to limit the number of features, we only evaluate the simpler approach Max Feature Set Size in the experiments.

Min Fairness. In many cases, it is imperative to train a model that avoids discriminating against any minority group, e.g. based on sex, race, or religion [63]. Fairness is difficult to define and there is no single metric that could capture all relevant aspects [43]. Developing measures for fairness is an active field of research. The goal of our work is to empower researchers and engineers to leverage the variety of existing fairness metrics. Fairness metrics, such as equal opportunity [29], the generalized entropy index [62], and the ratio of observational discrimination [56], have different perspectives on fairness but require all the same inputs - the dataset and the ML model. To exemplify how fairness can be embedded as a constraint, we chose the very simple and well-known fairness metric of equal opportunity (EO) [29] to measure fairness.

$$\begin{aligned}\text{EO} = 1 - |&P_{\text{minority group}}(Y_{\text{predicted}} = 1 | Y = 1) \\ &- P_{\text{majority group}}(Y_{\text{predicted}} = 1 | Y = 1)|\end{aligned}$$

EO considers the predictions as fair if the true positive rates for both the minority and the majority group are similar. Note that simply removing the sensitive or protected feature, such as race, is not enough to achieve fairness because other features might leak information about the same feature [63]. For instance, Selbst [58] showed that ZIP codes are a proxy for race in the U.S.

Min Privacy. With regulations, such as the General Data Protection Regulation (GDPR), the industry becomes more aware of focusing on the privacy of customers. ML models could leak information about customers whose data was used for model training. A popular model to preserve privacy is to apply ε-differential privacy for a classifier ϕ [20]. Differential privacy applies random noise to ensure that the publicly visible model predictions do not change significantly if one entry in the dataset changes. The user specifies ε as the constraint. To incorporate ε, one has to apply the differential private alternative of each ML model. For LR, we apply the differentially private empirical risk minimization [13]. For NB, we follow the approach of Vaidya et al. [66] and, for DT, we leverage work by Fletcher et al. [24]. By parameterizing the differentially private model with ε as a user constraint, one can expect that the output is ε-differential private.

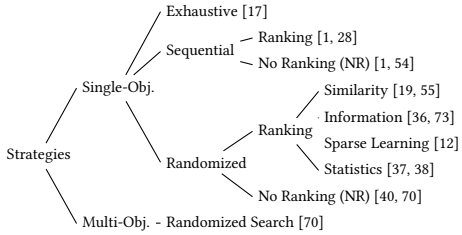

Figure 3: FS strategy taxonomy for Dfs.

Min Safety Against Adversarial Examples. Adversaries can attack ML models. For instance, adversaries can adjust the features of an instance in such a way that the instance is misclassified by the classifier. Again, there are many measures for safety against adversarial examples. We apply the following empirical robustness measure: For each test entry, we try to generate an adversarial example using the well-known black-box evasion attack HopSkipJump [15, 47]. Then, we compare the accuracy of the original test set with the accuracy of the attacked test set [48]: Safety $= 1 - F1(\text{Test}_{\text{original}}) - F1(\text{Test}_{\text{attacked}})$.

For Dfs, the user could also use other safety metrics, such as the clique method robustness [14], the cross Lipschitz extreme value for network robustness score [68], and loss sensitivity [5]. They all view robustness from a different angle but all require the same inputs. For the experiments, we want to study whether different FS strategies can satisfy the notion of safety against adversarial examples. Therefore, we chose the very simple and straightforward safety metric of empirical robustness.

4 FEATURE SELECTION

We first describe a taxonomy of FS strategies from the perspective of Dfs that guides to select the competing strategies. Then, we describe each selected FS strategy that we evaluate in our experiments.

4.1 Feature Selection Taxonomy

FS has been surveyed from multiple perspectives. Molina et al. [41] and Doak et al. [17] categorize FS strategies based on the search algorithm and the evaluation function. Li et al. [37] categorize the different feature ranking approaches into four categories: similarity-based, information-theoretical-based, sparse-learning-based, and statistical-based methods. Xue et al. [71] categorize the evolutionary computation approaches and introduce the new categorization dimension of the number of objectives because many evolutionary algorithms support multi-objective optimization by design.

We adopt a taxonomy in Figure 3 that describes the strategies from the perspective of Dfs. In this taxonomy, we classify FS strategies based on the number of optimization objectives, the underlying search algorithm, and whether they leverage a feature ranking. In contrast to previous surveys, we do not differentiate the strategies based on the evaluation function. For Dfs, we always have to evaluate the features based on the specified ML model - known as the wrapper approach [32] - to ensure that all user-specified constraints are satisfied. With the help of the taxonomy, we select the set of FS strategies for our experimental evaluation. We choose at least one FS strategy per taxonomy leaf for the experiments to get a broad representative picture of the strategies' performance for Dfs.

In Figure 3, we first differentiate FS strategies by the number of objectives that they optimize for. The user can specify a single constraint, such as accuracy, or multiple constraints, e.g. EO > 0.9 and accuracy > 0.8. One approach is to consider satisfying each constraint c_m individually as one objective in a multi-objective optimization problem. Another approach is to optimize how far away the feature subset is from satisfying all constraints together as one single aggregated objective.

Following previous surveys [17, 41], it is possible to then differentiate the strategies with respect to the search algorithm that they apply. Search can be exhaustive, sequential, and randomized.

In general, the task of searching for the optimal feature subset is known to be NP-hard [27]. So, if we leverage exhaustive search, assessing all possible feature subsets would require 2^N evaluations where N is the number of features. For large feature sets, it is intractable to use exhaustive search. Therefore, multiple heuristics have been proposed. Doak categorizes these heuristic strategies into sequential and randomized search strategies [17]. Sequential search incrementally adds or removes features for one feature subset. Randomized search picks features stochastically.

Sequential selection and single-objective randomized search can be further divided into strategies that use a ranking and those that do not (NR). For exhaustive search, one has to evaluate all possible feature subsets. Therefore, ranking does not apply to this type of strategies. For sequential search, recursive feature elimination [28] leverages the model's feature importance intuition as a ranking and prunes at each step the most insignificant feature. For randomized search, one can rank the features and pick the top-k. Randomized search will identify the optimal k. There have been a large number of rankings proposed that are based on similarity [19, 55], information theory [36, 73], sparse learning [12, 49], and statistics [37, 38].

Similarity-based strategies assess feature importance by measuring the distance between data instances. Information-theory-based strategies leverage the notion of information that is shared between the target and the features or among features. Sparse-learning-based strategies aim to learn an ML model that yields high accuracy while its weights are as sparse as possible. Finally, statistics-based strategies leverage well-known statistical measures, such as the variance [37] and the χ^2 score [38].

4.2 Feature Selection Strategies

We give a brief description of each FS strategy that we evaluate for Dfs. For all strategies, we specify the ranking they use or whether they use no ranking (NR). We can leverage all the described FS strategies for Dfs because they all follow the wrapper approach [32].

Exhaustive Search - ES(NR). ES(NR) evaluates all possible feature combinations.

Sequential Forward Selection - Sfs(NR). Sfs(NR) [1] starts with an empty feature set and then adds in each round the feature that benefits the current feature subset the most.

Sequential Backward Selection - Sbs(NR). In contrast to Sfs(NR), Sbs(NR) [1] starts with the full set of features and removes one feature per round to improve the feature subset. Both approaches Sfs(NR) and Sbs(NR) have a complexity of $O(N^2)$.

Sequential Forward Floating Selection - Sffs(NR). If it turns out that adding or removing a feature was a mistake, Pudil et al. [54] extend the sequential selection algorithms with floating. For forward selection, floating checks after adding a new feature whether removing any feature from the current set is beneficial.

Sequential Backward Floating Selection - Sbfs(NR). For backward selection, the floating approach of Sbs(NR) would try to add back previously removed features [54].

Recursive Feature Elimination - Rfe(Model). Rfe(Model) is another well-known sequential selection strategy [28]. It uses backward selection but instead of choosing the feature that should be removed, using the wrapper approach, it leverages a feature ranking function, e.g., the feature importance scores of the classification model at hand. If the classification model does not provide feature importance scores, we estimate these scores using the permutation importance [11]. Another way to leverage rankings for FS is to pick the top-k features. Approaches based on the second category randomized search, better known as HPO, find the optimal value for k. We choose the well-known tree-structured Parzen estimator approach (Tpe)[7] for this task. To reduce the computation, we compute each ranking only once in the first round of HPO.

Top-k ReliefF Selection - Tpe(ReliefF). The main idea of similarity-based Relief algorithms is to choose a random instance and find its nearest neighbor with the same class (near hit) and its nearest neighbor with a different class (near miss). Based on these neighbors, one can incrementally compute a feature ranking by continuously drawing new instances. ReliefF [55] extends this idea by selecting the k-nearest neighbors instead of only one.

Top-k Fisher Score Selection - Tpe(Fisher). The Fisher score ranking [19] also ranks features in a way that feature values of instances of the same class are similar while feature values of instances of the different classes are dissimilar. This way, the features with the highest Fisher score are ranked highest.

Top-k Mutual Information Maximization Selection - Tpe(Mim). Information-theory-based Mim [36] ranks a feature X based on how much information it shares with the target Y by calculating the mutual information. Mim does not prune redundant features because it assumes that features are independent.

Top-k Fast Correlation-Based Filtering - Tpe(Fcbf). Information-theory-based Fcbf [73] considers both feature-target correlations and feature-feature correlations. Fcbf compensates for mutual information's bias towards features with more values by computing the symmetrical uncertainty (SU) [73].

Top-k Multi-Cluster FS - Tpe(Mcfs). The sparse-learning-based Mcfs [12] does not require class labels and tries to compress the data by selecting as few features as possible while maintaining the local geometric structure. First, it transforms the data into a K-dimensional spectral embedding $e_1, ..., e_k$ [46]. Then, it considers each of the resulting k dimensions as a target and the original data as the features. This way, one can formulate it as K regression tasks.

Top-k Variance Selection - Tpe(Variance). The intuition behind the statistics-based strategy Tpe(Variance) is that features with low variance contain less information [37] and therefore, are less likely to help to differentiate between different classes.

Top-k χ^2 Score Selection - Tpe(χ^2). The ranking that uses the χ^2 score tests whether the feature f is independent of the class label [38]. Higher χ^2 score signalizes higher feature importance.

Simulated Annealing - SA(NR). To use randomized search without feature ranking, one can consider each decision whether to use

a feature as a binary variable where 1 corresponds to select a feature and 0 to not select it. For instance, the binary vector $b = (0, 1, 1)$ corresponds to the decision to prune the first feature of a 3-feature subset. Therefore, the optimization goal is to find the binary vector b that optimizes a given objective. To solve this optimization problem, we can leverage simulated annealing [17].

Tree-Structured Parzen Estimator - TPE(NR). We can solve the same optimization problem of SA(NR) also with the well-known HPO tree-structured Parzen estimator approach [7].

Nondominated Sorting Genetic Algorithm II - NSGA-II(NR). For multiple objectives, Xue et al. [71] follow the same optimization approach as SA(NR) and TPE(NR) but instead of finding the best feature subset for one objective, they want it to be optimal across multiple objectives. Evolutionary algorithms, such as the well-known nondominated sorting genetic algorithm II (NSGA-II) [71], represent one of the most well-known optimization algorithm families. Therefore, we treat each constraint as one objective. For instance, the constraint set accuracy > 80% and EO > 90% is translated into two objectives: one for accuracy and one for fairness.

4.3 Guiding FS to Satisfy Constraints

To enable the single-objective strategies to find feature sets that satisfy multiple constraints, we aggregate the distance to each constraint threshold in a single objective function. Our goal is to minimize the overall distance across all constraints. Therefore, we sum up the squared distance of each achieved validation score δ_m to its corresponding constraint threshold:

$$\text{distance} = \sum_{m=1}^{M} \begin{cases} 0, & \text{if } c_m \text{ is satisfied} \\ (\delta_m - c_m)^2, & \text{otherwise.} \end{cases} \quad (1)$$

So, instead of optimizing for classification accuracy, we optimize for minimal distance to satisfy the constraints. We treat all constraints equally because all constraints can take values between 0 and 1 and all constraints should be satisfied equally in the end. Note that differential privacy constraints are already satisfied before optimization and are therefore not part of the optimization. If the user wants to maximize utility, e.g., accuracy, subject to specified constraints being met, we extend the objective function of Equation 1:

$$\text{objective} = \begin{cases} \text{distance}, & \text{if distance} > 0 \\ \sum_{n=1}^{N} -u_n, & \text{otherwise.} \end{cases} \quad (2)$$

Once all constraints c_m are satisfied (distance = 0), we continue the optimization by minimizing the sum over all defined negative utilities until the maximum search time is reached.

5 DFS OPTIMIZER

Given the set of FS strategies, there is a selection problem for a given dataset, a given classification model, and a set of desired constraints. One simple option would be to always propose the FS strategy that was the fastest or satisfied the most constraint sets on a large number of benchmark tasks. However, as the experiments show, there is no one-size-fits-all FS strategy that is always the fastest or can satisfy all possible constraint sets. Therefore, we explored the possibility for a DFS optimizer based on meta-learning that predicts the FS strategy that is the most likely to satisfy the user-specified constraints - the query.

The main requirements for such an optimizer are the following: First, it should identify a promising FS strategy without testing them on the given dataset. Second, the optimizer should consider the user-specified constraints and should be easily extensible to more constraints and more FS strategies. To accommodate the above-mentioned requirements, we have to design a classification task that can convert the accessible information of an ML scenario into a feature representation and obtain the necessary training data. To the best of our knowledge, the DFS optimizer is the first technical solution to accommodate the aforementioned requirements and challenges to automatically choose the proper FS algorithm for a given ML scenario.

Algorithm 1 DFS Optimizer

Training Phase

Input: datasets $\mathbb{D}$, models Φ, hyperparameters H, FS strategies S, constraints C, iterations I.
Output: model_s.

$X \leftarrow [\,]$
for $i \leftarrow 0$ to I **do**
  $D, \phi, C' \leftarrow \text{sample}(\mathbb{D}, \Phi, C)$
  $X \leftarrow X \cup \text{featurize}(D, \phi, C')$
  for s *in* S **do**
    $y_s \leftarrow y_s \cup \text{strategy\_satisfies}(s, D, \phi, C', H)$
for s *in* S **do**
  $\text{model}_s \leftarrow \text{fit}(X, y_s)$

Deployment Phase

Input: dataset $\widetilde{D}$, model $\widetilde{\phi}$, constraints $\widetilde{C}$.
Output: feature selection strategy $\widetilde{s}$.

$\widetilde{x} \leftarrow \text{featurize}(\widetilde{D}, \widetilde{\phi}, \widetilde{C})$
$\widetilde{s} \leftarrow \arg\max_s \text{model}_s.\text{predict\_proba}(\widetilde{x})$

5.1 Algorithm Overview

We formulate the meta-learning problem as a multi-label binary classification task where we predict for each strategy whether it can satisfy a given ML scenario or not. Algorithm 1 describes how DFS Optimizer is trained and deployed. In the training phase, we train one model for each strategy to estimate the probability of the corresponding strategy to satisfy the ML scenario. For this purpose, previously deployed ML scenarios can serve as training data. From each scenario, we have access to the constraints, models with their optimized hyperparameters, and datasets, to extract features. In absence of such training data, we can automatically generate the training data as depicted in Algorithm 1. For the specified number of iterations I, which bounds the sample size, we sample an ML scenario (a dataset D, a model ϕ, a constraint set C) and verify for each FS strategy s whether it can satisfy the scenario or not. Based on these evaluations, we create a training dataset that includes the observations X, consisting of the features for each ML scenario that we describe in more detail in Section 5.2, and the target y_s, which describes for each ML scenario whether strategy s could satisfy it.

We use the training dataset to train one model per FS strategy to estimate the probability of the corresponding strategy to satisfy the ML scenario. In the deployment phase, the DFS Optimizer then estimates for each strategy s the probability of success on a user-provided ML scenario (a dataset $\widetilde{D}$, a model $\widetilde{\phi}$, a constraint set $\widetilde{C}$) and chooses the strategy with the highest probability.

5.2 ML Scenario Representation

To enable the aforementioned approach, we need a feature vector that covers four components: the dataset at hand, the classification model, the constraints, and the potential hardness of the constraints:

$$\rho(D, \phi, C) = [\rho_{\text{data}}(D), \rho_{\text{model}}(\phi), \rho_{\text{constraints}}(C), \rho_{\text{hardness}}(D, \phi, C)].$$

Dataset features. We encode the dataset with its number of instances and features: $\rho_{\text{data}}(D) = [\text{rows}(D), \text{features}(D)]$. The experiments in Section 6.3 show that some of the FS strategies do not scale well either for the number of instances or features.

Model features. We one-hot encode the classification model as $\rho_{\text{model}}(\phi) = \bigcup_{\widetilde{\phi}}^{\Phi} \phi == \widetilde{\phi}$, where Φ is the list of all classification models that have been used in training.

Constraint features. To encode the constraints, we need to capture two types of information. The intuition here is that the model will have a better estimation of the performance of a selection strategy, first if the selected user constraints are similar to those of a training ML scenario, and second if the possibility of satisfying the given constraints is similar.

Encoding the former is straightforward. We simply encode all user-provided constraints (in our benchmark 6 constraints) in a feature vector: $\rho_{\text{constraints}}(C) = \bigcup_{c}^{C} c$. All these constraints might affect the choice of the FS strategy. For instance, a backward selection strategy might require too much time to reach feature subsets that require very few features or a compute-intensive ranking-based strategy might require more time than the specified maximum search time. By encoding the maximum search time constraint as a feature, the model will predict rather fast selection strategies. Note that the optimizer can be extended to any custom constraint.

To capture how likely it is that a given constraint set can be satisfied for the current ML scenario, we design a feature based on the intuition that the distance of the measured metrics on the original dataset to the desired constraint thresholds can serve as priors for this purpose. However, if the dataset has many instances, computing these distances for the full dataset is time-intensive. Instead of evaluating the full dataset, we evaluate all features for a small class-stratified sample using cross-validation. This approach is called subsampling-based landmarking [25]. So, we encode the ML task by adding the difference of the 6 constraints with their corresponding cross-validation metric cv_c estimate to the feature vector: $\rho_{\text{hardness}}(D, \phi, C) = \bigcup_{c}^{C} cv_{c(D,\phi)} - c$.

The described solution ensures fast deployment because it requires only inference and landmarking to answer a query. Second, it is extendable both for constraints and FS strategies. For a new constraint, we simply add one more feature to the vector. For a new FS strategy, we simply train one more classification model. So, in case of a new constraint or FS strategy, the system would generate a large number of ML scenarios for these new options and execute them. Then, it would retrain the optimizer.

6 EXPERIMENTS

We performed extensive experiments to answer the following questions: (1) Which FS strategy can cover the most ML scenarios? (2) Which FS strategy is the fastest in satisfying an ML scenario? (3) How does HPO affect FS? (4) Can all FS strategies satisfy constraints based on novel metrics, such as EO and safety? (5) How does the choice of the model affect FS? (6) Can we combine FS strategies to yield faster results or higher coverage? (7) Does the Dfs optimizer reach a higher coverage than the single strategies?

Table 2: Experimental datasets.

Dataset	Instances	Attributes	Features	Sensitive Attribute
Traffic Violations	1578154	34	2075	Race
AirlinesCodrnaAdult	1076790	30	746	Gender
Adult	48842	15	108	Gender
KDD Internet Usage	10108	69	526	Gender
IPUMS Census	8844	57	274	Gender
Telco Customer Churn	7043	20	45	Gender
COMPAS	5278	14	19	Race
Students	3892	35	39	Gender
Thyroid Disease	3772	30	54	Gender
Primary Biliary Cirrhosis	1945	19	723	Gender
Titanic	1309	12	422	Gender
Social Mobility	1156	6	39	Race
German Credit	1000	21	61	Nationality
Indian Liver Patient	583	11	11	Gender
Irish Educational Transitions	500	6	18	Gender
Arrhythmia	452	280	334	Gender
Brazil Tourism	412	9	22	Gender
Primary Tumor	339	18	41	Gender
Diabetic Mellitus	281	98	98	Gender

6.1 Evaluation Metrics

To show the effectiveness of different Dfs strategies, we are interested in two main characteristics. First, how many of the ML scenarios can a given strategy cover? Second, how many times is a given strategy the fastest in satisfying a given ML scenario? We calculate the coverage for the strategy s and the set of ML scenarios $\mathbb{Z}$ as follows: $\text{coverage}(s, \mathbb{Z}) = \frac{|\{Z | Z \in \mathbb{Z} \wedge s(Z) \neq \emptyset\}|}{|\mathbb{Z}|}$. For each ML scenario Z, we count how often the strategy s finds a solution and does not yield the empty set. To compute the coverage, we divide this number by the total number of ML scenarios. Note that the maximum search time is a mandatory constraint. So, the strategies have to find a solution within the corresponding specified time limit. To show the speed of different Dfs strategies, we report the fraction of ML scenarios where a given strategy yields a result the fastest, denoted as *Fastest*. To evaluate the effectiveness of our Dfs optimizer, we follow the leave-one-out cross-validation approach by always considering the experiments of one dataset as the test set.

For each FS strategy, we would like to evaluate the performance for a given constraint set on a given dataset with a given classification model. However, the space of constraints is infinite. Even if we would discretize each constraint into v values, for 6 constraints, 19 datasets, $v = 5$, 16 strategies, 3 classification models, 1 hour maximum search time, in the worst case, it would take $|\text{constraints}|^v * |\text{datasets}| * |\text{models}| * \text{max search time} = 51$ years for the experiments of one single strategy.

To estimate the performance for the entire query space, Seltenreich et al. [59] propose to leverage domain-aware randomized fuzzing. Following this approach, we pick a random constraint set and let all described strategies search for features that satisfy this constraint set on a randomly picked dataset. We generated three versions of this benchmark - undergoing four weeks of pure computation time:

(1) with default model hyperparameters (1500 ML scenarios),
(2) with model HPO (3318 ML scenarios),
(3) and considering F1 as utility subject to the other specified constraints (957 ML scenarios) - also includes HPO.

For HPO, we optimize for F1 score using grid search. For LR, we optimize the regularization strength $C \in \{10^n \| n \in [-2:3]\}$. For NB, we optimize the smoothing variable var_smoothing $\in [1e-12 : 1e-6]$. For DT, we optimize the maximum tree depth $td \in [1:7]$.

Listing 1 shows the template that we use to sample from the ML scenario space. First, we sample the classification model. Then, we sample the constraints. There are mandatory and optional constraints. Minimum accuracy and maximum search time are mandatory. We sample the minimum F1 score uniformly in the range between 0.5 and 1. We do not sample values lower than 0.5 because no user is interested in features that yield a model that is less accurate than random guessing. For the maximum search time, we uniformly sample between 10 seconds and 3 hours. Note that coverage is strictly bound by the maximum allowed search time. Theoretically, one could make the sampling of maximum search time data dependent. However, before evaluating a set of constraints, there is no way of knowing the runtime. Approaches could require only 1 minute or not terminate at all. For the sake of feasibility, we limited the maximum amount of time to 3 hours.

Listing 1: Constraint space template.

```
from hyperopt import hp
constraint_space = {
'classifier':   hp.choice('LR', 'DT', 'NB']),
'min_f1':   hp.uniform('val', 0.5, 1),
'max_time':  hp.uniform('val', 10, 3 * 60 * 60),
'max_features':   hp.choice('?', [1, hp.uniform('val', 0, 1)]),
'min_EO':   hp.choice('?', [0, hp.uniform('val', 0.8, 1)]),
'min_safety':  hp.choice('?', [0, hp.uniform('val', 0.8, 1)]),
'privacy_ε': hp.choice('?', [None, hp.lognormal('val', 0, 1)])}
```

For optional constraints, we randomly choose whether to use the constraint or not. For the maximum feature fraction, we uniformly sample a value between 0 and 1. For EO and safety, we sample only between values of 0.8 and 1 because, for instance, if a user is interested in enforcing fairness, she won't be interested in setting a constraint where the groups differ more than 20%. The same applies to safety. For differential privacy, we pick ε based on the log-normal distribution with $\mu = 0$ and $\sigma = 1$. In our evaluation, we focus on the ML scenarios where at least one FS strategy found a feature set that satisfied all constraints. However, we do provide an analysis of the cases that could not be satisfied by any FS strategy in Section 6.3.

Datasets. We chose the binary classification datasets based on whether they are openly accessible, e.g. on OpenML [67], and whether they include a sensitive attribute that can be considered for fairness. This way, we found 19 datasets that satisfied these requirements. Dfs is independent of the ML task and can be also used for regression and clustering. We focus on classification in our experimental study, which allows us to have a comparable set of metric implementations. The datasets vary with respect to the number of numeric features, categorical features, and missing values. We removed textual attributes because we focus on categorical and numerical features in this study. The adjusted datasets can be found in our repository [45]. Table 2 lists these datasets with basic statistics and the sensitive attribute that we consider for the fairness metric equal opportunity. Using stratification, we split each dataset into a training, validation, and test set in a 3:1:1 ratio. For each dataset, we apply standard preprocessing transformations such as one-hot encoding for all categorical attributes. For all numerical attributes, we apply min-max scaling and mean value imputation. Scaling accelerates the convergence of LR and imputation removes any missing values that many ML models cannot process. We choose this preprocessing pipeline because it is standard, simple, and does not affect the interpretability of the resulting model. For instance, if we would choose to apply feature hashing [42] or principal component analysis [50], the meaning of the features would be lost in the corresponding embedding. In this paper, we focus on FS and leave feature construction as future work.

Table 3: Fraction of Fastest cases and coverage per strategy

Strategy	Default Parameters		Parameter Optimization	
Strategy	Fastest Fraction	Coverage Fraction	Fastest Fract.	Coverage Fract.
Original Features	0.01 ± 0.04	0.14 ± 0.10	0.05 ± 0.04	0.21 ± 0.13
SBS(NR)	0.00 ± 0.01	0.27 ± 0.17	0.02 ± 0.03	0.28 ± 0.17
SBFS(NR)	0.03 ± 0.05	0.27 ± 0.15	0.03 ± 0.04	0.28 ± 0.16
RFE(Model)	0.01 ± 0.03	0.29 ± 0.16	0.02 ± 0.03	0.37 ± 0.19
TPE(MCFS)	0.01 ± 0.03	0.34 ± 0.21	0.01 ± 0.03	0.38 ± 0.23
TPE(ReliefF)	0.02 ± 0.05	0.47 ± 0.24	0.02 ± 0.03	0.48 ± 0.23
TPE(Variance)	**0.15 ± 0.15**	0.49 ± 0.22	0.06 ± 0.07	0.48 ± 0.21
TPE(NR)	0.05 ± 0.06	0.50 ± 0.17	0.07 ± 0.07	0.49 ± 0.20
NSGA-II(NR)	0.08 ± 0.08	0.53 ± 0.16	0.08 ± 0.06	0.49 ± 0.19
TPE(MIM)	0.04 ± 0.05	0.49 ± 0.23	0.04 ± 0.05	0.53 ± 0.26
SA(NR)	0.06 ± 0.07	0.49 ± 0.19	0.07 ± 0.04	0.54 ± 0.17
ES(NR)	0.10 ± 0.09	0.55 ± 0.27	0.11 ± 0.10	0.55 ± 0.31
TPE(Fisher)	0.06 ± 0.06	0.49 ± 0.25	0.04 ± 0.05	0.56 ± 0.24
TPE(χ^2)	0.10 ± 0.08	0.53 ± 0.19	0.06 ± 0.06	0.57 ± 0.21
SFS(NR)	0.09 ± 0.10	0.60 ± 0.25	0.10 ± 0.10	0.58 ± 0.30
SFFS(NR)	0.10 ± 0.08	**0.62 ± 0.22**	**0.12 ± 0.12**	0.59 ± 0.31
TPE(FCBF)	0.09 ± 0.11	0.50 ± 0.20	0.11 ± 0.22	**0.60 ± 0.22**
DFS Optimizer	0.14 ± 0.14	**0.65 ± 0.14**	0.11 ± 0.05	**0.70 ± 0.18**
Oracle	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00

6.2 Usage of Strategies

Our benchmark comprises 16 FS strategies that we described in Section 4. We provide all these strategies in our repository [45]. We implemented ES(Nr), Sfs(Nr), Sffs(Nr), Sbs(Nr), Sbfs(Nr), and Rfe(Model). We computed the Fisher score, Fcbf, and Mcfs rankings using the implementation provided by Li et al. [37]. For ReliefF, we leveraged the implementation by Urbanowicz et al. [65] and used the default number of nearest neighbors of 10. For Mim and the χ^2 score, we leverage the Scikit-learn library [51].

For the FS strategies that optimize the binary decision vector, Bergstra et al. [7] implemented SA and Tpe. Blank et al. [8] implemented NSGA-II. We follow the configuration by Xue et al. [70] to set the NSGA-II parameter population size to 30.

As the classification model for the optimizer, we choose a random forest classifier with default parameters and class balancing. For subsampling-based landmarking, we choose a sampling size of 100 instances which correspond to the size of the smallest training set in our benchmark. We run the experiments on Ubuntu 16.04 machines with 28 2.60 GHz cores and 264 GB memory.

6.3 Strategy Effectiveness

We evaluate the strategies' performance first aggregated across all datasets and then fine-grained by datasets, constraint types, classification models, and finally analyze which strategies come closest to satisfy constraints even if they fail.

Aggregated Effectiveness. Table 3 reports statistics for coverage and comparative runtime of each FS with and without model HPO. It reports the average coverage fraction of satisfiable ML scenarios per strategy. It also shows the average fraction of cases where a strategy was fastest to cover a scenario. At a first glance, none of the

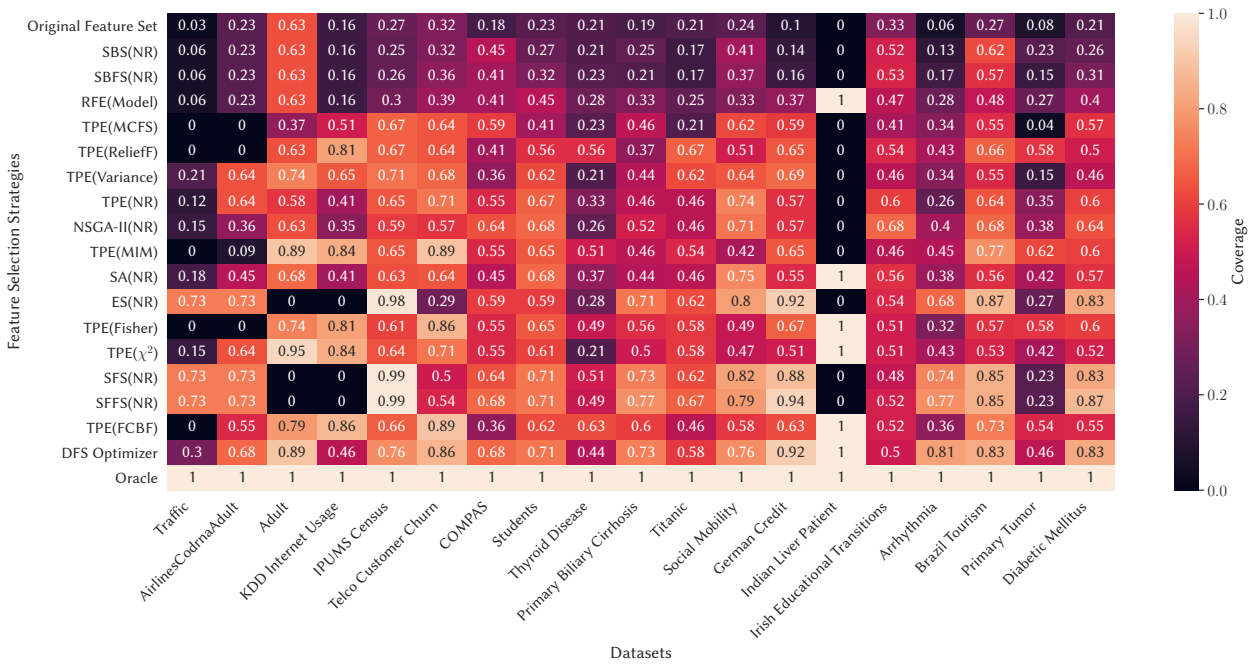

Figure 4: Strategies' coverage for individual datasets.

FS strategies reaches 100% coverage. We also report the result for using the original complete feature set without FS. Without HPO, only 14% of the ML scenarios can be covered with this feature set because most constraints, in particular Min EO, and Min Privacy, require a smaller subset of features. With HPO, the fraction slightly increases because optimization helps to achieve a higher F1 score.

HPO. In general, we see that in most cases the coverage between the strategies under default parameters and under HPO differs only by a few percentage points. For FS strategies that require only few model evaluations, HPO leads to higher coverage. Ranking strategies, such as Tpe(Fcbf) and Tpe(Fisher), clearly benefit from HPO. Rfe(Model)'s coverage increases by 8% because it bases its ranking on the model's scores. If the model can fit the data better, the ranking fits the data better and therefore leads to higher coverage. FS strategies that require more evaluations, such as Sffs and Sfs, lose a few percentage points in coverage. As most selection strategies benefit from HPO and most users expect the model to be tuned, we report for all further experiments the results based on HPO.

Coverage. Ranking strategies that provide a concise and lightweight ranking, such as Tpe(Fcbf), Tpe(χ^2), and Tpe(Fisher) reach a coverage higher than 50%. The forward selection strategies Sfs and Sffs also achieve very high coverage of 58% and 59%, respectively. Most constraints, such as min privacy, max feature set size, and min robustness, require a small set of features. For the same reason, backward selection strategies, such as Sbs(Nr), Sbfs(Nr), and Rfe(Model), show poor coverage because, for small feature subsets, they have a high runtime that exceeds the specified maximum search time. In particular, they are slower than other approaches because they do not benefit from the optimizations based on the maximum feature set size. Our experiments confirm Pudil et al.s' finding [54] that the floating alternatives Sffs and Sbfs of the sequential FS approaches Sfs and Sbs, respectively, provide more optimal solutions. The selection strategies that operate on a binary feature decision vector, such as Tpe(Nr), SA(Nr), and NSGA-II(Nr), perform similarly and solve 49% - 54% of the ML scenarios. Mcfs shows a poorer coverage of 38% because of the time-intensive computation of the spectral embedding.

Fastest. Table 3 also reports that no strategy is always the *fastest* in finding a satisfying feature set. The best strategy with respect to finishing fastest is Sffs(Nr), which is the fastest strategy in 12% of the ML scenarios on average because it starts with very few features. Therefore, training a model takes less time in the beginning. Further, Sffs(Nr) can greedily find feature sets that satisfy challenging constraints, such as fairness or safety. In case that constraints require a large number of features, ranking strategies, such as Tpe(Fcbf), are the fastest.

Oracle. Table 3 contains also the results of an oracle that always chooses the fastest strategy for a given ML scenario. Therefore, it satisfies all possible ML scenarios and achieves 100% coverage. Since the best single strategy Tpe(Fcbf) achieves only 60% with a large standard deviation of 22%, there is a huge potential of choosing the right strategy depending on the ML scenario. In Section 6.6, we further explore this direction by evaluating our Dfs optimizer.

Dataset-Specific Effectiveness. Figure 4 provides the strategies' performance broken down to the dataset level. Indeed, the strategies' performance is dataset dependent. In particular, the dimensions of a dataset affect the strategies' performance. For instance, the ranking computation of Mcfs, Relieff, Fisher, Mim, and Fcbf are not scalable and therefore, find no feature sets on Traffic, which has the highest number of instances. For the datasets with a large

Table 4: Distance to constraints for unsuccessful cases and the achieved F1 score for the utility-driven benchmark.

Strategy	Distance for Validation	Distance for Test	Mean Normalized F1 score
Original Feature Set	0.43 ± 0.44	0.44 ± 0.45	0.16 ± 0.09
SBS(NR)	0.31 ± 0.44	0.38 ± 0.44	0.36 ± 0.22
SBFS(NR)	0.31 ± 0.44	0.38 ± 0.44	0.36 ± 0.23
RFE(Model)	0.29 ± 0.43	0.34 ± 0.43	0.30 ± 0.15
TPE(MCFS)	0.36 ± 0.63	0.39 ± 0.60	0.46 ± 0.22
TPE(ReliefF)	0.32 ± 0.62	0.35 ± 0.59	0.43 ± 0.24
TPE(Variance)	0.21 ± 0.41	0.25 ± 0.40	0.48 ± 0.22
TPE(NR)	0.18 ± 0.42	0.25 ± 0.41	0.62 ± 0.22
NSGA-II(NR)	0.19 ± 0.42	0.26 ± 0.41	0.62 ± 0.23
TPE(MIM)	0.27 ± 0.55	0.31 ± 0.52	0.45 ± 0.23
SA(NR)	0.19 ± 0.43	0.25 ± 0.41	0.63 ± 0.21
ES(NR)	0.16 ± 0.30	**0.20 ± 0.31**	0.73 ± 0.17
TPE(Fisher)	0.31 ± 0.59	0.33 ± 0.56	0.43 ± 0.24
TPE(χ^2)	0.20 ± 0.40	0.24 ± 0.40	0.48 ± 0.20
SFS(NR)	**0.15 ± 0.30**	**0.20 ± 0.31**	0.75 ± 0.17
SFFS(NR)	**0.15 ± 0.30**	**0.20 ± 0.31**	**0.77 ± 0.16**
TPE(FCBF)	0.22 ± 0.45	0.26 ± 0.44	0.49 ± 0.23

number of features, such as Traffic and KDD Internet Usage, backward selection strategies, such as Sbs, Sbfs, and Rfe, perform poorly because they require more evaluations to reach feature sets with few features. Furthermore, the information distribution across features affects the strategies' performance. For instance, for datasets with few critical features, such as IPUMS Census, COMPAS, Titanic, and German Credit, the forward selection approaches Sfs and Sffs achieve the highest coverage. For the other datasets, the ranking-based strategies Tpe(Fcbf), Tpe(χ^2), and Tpe(Fisher) achieve the highest coverage. For instance, χ^2 works better in the predominantly categorical dataset Adult.

Optimizing for Utility. In some cases, the user might not have constraints for all dimensions and rather wants to optimize for some utility subject to certain constraints. We conduct another extensive benchmark and specify the F1 score as utility. To report a score that is comparable across ML scenarios i and datasets d, we compute the normalized mean F1 score for each FS strategy s:

$$\text{normalized mean F1 score}(s) = \sum_{d}^{D}(\sum_{i}^{I} F1(d,i,s)/max(F1(d,i))/I)/D$$

As for some ML scenarios, e.g. specified constraints, it is easier to achieve a high F1 score than for other ML scenarios, for each ML scenario, we divide the achieved F1 score of a given strategy by the best F1 score that was achieved by any FS strategy. As the datasets also differ in difficulty, we calculate the mean normalized F1 score per dataset and then report the mean across datasets as shown in Table 4. Sffs(Nr) achieves the highest normalized F1 score. It fully utilizes the given maximum search time for a larger space of representations and outperforms Tpe(Fcbf), which achieved the best coverage for the constraint satisfaction use case.

Failure Analysis. To understand the ML scenarios where all strategies fail, we compare the value distributions for the constraint sets where Dfs succeeds and where it fails. The comparison shows that the failed ML scenarios require significantly higher accuracy, EO, and privacy thresholds compared to the successful scenarios. For example, for the Adult dataset, Dfs could not satisfy any ML scenario with a minimum F1 score constraint higher than 63%. These thresholds are again dataset dependent.

Furthermore, we analyzed whether the maximum search time is the issue of why the strategies cannot find a solution. The sequential forward selection strategy Sfs(Nr) finishes in 22% of the failed scenarios, Tpe(χ^2) finishes even in 62% of the cases, and exhaustive search finishes in 14% of the cases. So, at least in 14% of the cases, the constraints were impossible to satisfy because no possible feature combination could satisfy the given ML scenario.

Table 5: The coverage if a constraint was specified.

Strategy	Min EO	Max Feature Set Size	Min Safety	Min Privacy
Original Feature Set	0.29	0.00	0.00	0.11
SBS(NR)	0.29	0.00	0.00	0.22
SBFS(NR)	0.29	0.00	0.00	0.22
RFE(Model)	0.14	0.14	0.00	0.11
TPE(MCFS)	0.57	0.14	0.17	0.33
TPE(ReliefF)	0.29	0.29	0.00	0.11
TPE(Variance)	0.57	0.29	0.17	0.44
TPE(NR)	0.43	0.43	0.33	0.22
NSGA-II(NR)	0.43	0.43	0.17	0.33
TPE(MIM)	0.43	0.43	0.00	0.22
SA(NR)	0.43	0.43	0.17	0.11
ES(NR)	0.71	0.43	0.50	0.56
TPE(Fisher)	0.29	0.43	0.00	0.22
TPE(χ^2)	0.29	0.29	0.00	0.22
SFS(NR)	**0.71**	0.43	0.67	0.67
SFFS(NR)	**0.71**	**0.57**	**0.83**	**0.78**
TPE(FCBF)	0.43	0.43	0.17	0.22

Table 6: Model-dependent coverage

Strategy	LR	NB	DT
Original Feature Set	0.22	0.12	0.18
SBS(NR)	0.29	0.16	0.26
SBFS(NR)	0.29	0.16	0.25
RFE(Model)	0.44	0.16	0.27
TPE(MCFS)	0.39	0.29	0.32
TPE(ReliefF)	0.46	0.43	0.36
TPE(Variance)	0.46	0.40	0.38
TPE(NR)	0.51	0.32	0.42
NSGA-II(NR)	0.53	0.31	0.41
TPE(MIM)	0.52	0.43	0.42
SA(NR)	0.59	0.30	0.40
ES(NR)	0.46	0.46	0.47
TPE(Fisher)	0.56	0.41	0.39
TPE(χ^2)	0.55	0.42	0.40
SFS(NR)	0.47	0.48	0.50
SFFS(NR)	0.48	**0.49**	**0.52**
TPE(FCBF)	**0.60**	0.41	0.45

For the failed cases, we also analyze how close the approaches got to the thresholds in Table 4. We report the average distance to the constraints as defined in Equation 1 on the validation set and test set for the failed cases. Again, forward selection strategies, such as Sfs(Nr) and Sffs(Nr), outperform the other strategies and come closest to solving the constraints on average.

Constraint-Specific Effectiveness. The strategies' performance is not only highly dependent on the dataset at hand but also on the constraint set at hand. To understand which constraints are harder or easier to satisfy, we report the aggregated coverage of scenarios that had one optional constraint in common. Min accuracy and max search time are mandatory constraints and always specified. Table 5 shows that the forward selection strategies Sffs(Nr) and Sfs(Nr) outperform all other strategies for the different constraint types. For safety and privacy constraints, Sffs and Sfs achieve significantly higher coverage than the other strategies. The reason is that these constraint types require very small feature sets that forward selection strategies uncover the fastest. However, it is surprising that accuracy-optimized ranking-based FS strategies, such as Tpe(Fcbf), can satisfy constraints based on novel metrics, such as safety, EO, and privacy, to a similar degree as the highly flexible multi-objective NSGA-II(Nr) strategy. One reason is that flexibility introduces the danger of overfitting. Another reason is that the ranking priors that are designed for accuracy, e.g. mutual information, can be also used to prune biased features or reduce the feature set for privacy reasons.

Model-Specific Effectiveness. The strategies' performance is also affected by the classification model that is chosen. Table 6 shows the test coverage across the three different classification

Table 7: Percentage of feature sets found using LR that satisfy constraints under a DT, a NB, and a SVM model.

Strategy	Min Accuracy	Min EO	Min Safety
DT (SFFS)	0.93 ± 0.18	0.95 ± 0.14	0.63 ± 0.34
NB (SFFS)	0.85 ± 0.30	0.79 ± 0.34	0.67 ± 0.38
SVM (SFFS)	0.90 ± 0.26	0.81 ± 0.32	0.88 ± 0.25

models LR, NB, and DT. While for all three models, forward selection Sffs leads to high coverage, TPE(Fcbf) achieves the highest coverage for LR because both Fcbf and LR assume linearity of independent variables. The same argument applies to Tpe(χ^2).

Now, we describe two other notable differences among models. First, NB achieves significantly lower coverage for Rfe than the other models. The reason is that NB does not provide a notion of feature importance and therefore, we have to additionally calculate the feature ranking using feature permutation importance that leads to significant runtime overhead that might exceed the specified maximum search time. Second, LR performs better for strategies that operate on a binary feature decision vector, such as Tpe(Nr), SA(Nr), and NSGA-II(Nr), because they benefit from more evaluations and LR is the least computationally intensive model.

Reusability of Feature Sets across Models. As we enforce the constraints on the feature level, another interesting question is whether the constraints hold also when we train a different model on the same feature set. To ensure that this is true, instead of evaluating the features using only one model, we could always use multiple models. However, this would require a computational overhead. In this experiment, we test whether features that were found by an FS strategy for an LR model are also useful for other models, such as DT, NB, and SVM models. We evaluated whether the constraints were still satisfied or not. The results of this analysis are presented in Table 7. As some constraints, such as the number of features are inherently model-independent, we focused this study on accuracy, EO, and safety constraints. Due to space limitations, we choose Sffs(Nr) because it showed the highest coverage across constraints (Table 5). The other numbers can be found online [45].

In general, for the majority of the ML scenarios, the constraints obtained through LR also hold for other models. The result suggests that, in typical ML development cycles, it is possible to change the model without searching again for features that satisfy constraints. This can be useful in highly iterative development pipelines. However, there are small differences across constraints. For instance, safety against adversarial examples is more model dependent than accuracy and EO. Thus, the resulted number of satisfied ML scenarios was comparably lower.

6.4 Impact of Constraint Type

We analyze the impact of constraints with the following constraint pairs: accuracy×{ EO, privacy, number of features, safety against adversarial examples }. For all these constraint pairs, we apply all strategies for all combinations in a specified grid on the Adult dataset and report the fastest strategy of 5 runs in Figure 5. The omitted areas inside each chart were covered by the adjacently covered approaches.

First, we see that strategy performance varies across charts, i.e., constraint types, and within charts, i.e., constraint thresholds. For safety constraints, Tpe(Variance) is the best choice for the Adult dataset. Selecting the features with the highest entropy seems to reduce the success of adversarial attacks. For feature set size and privacy constraints, Tpe(Variance) is the fastest for less restrictive constraint sets, i.e. F1 score < 0.59. However, for more restrictive constraints sets (upper right in the charts), i.e. the respective Pareto front, ranking strategies with more advanced priors are better. For instance, Tpe(χ^2) considers the relationship to the target class, Tpe(Fcbf) additionally considers redundancies among features, and Tpe(Mim) considers the information that is shared between features and the target. We see that Tpe(Variance) is the fastest for EO < 0.81. Higher EO constraints are predominantly satisfied by approaches, such as Tpe(Nr) and SA(Nr), that can prune specific biased features and are not bound to a ranking.

In conclusion, ranking-based approaches, such as Tpe(χ^2), Tpe(Fcbf), and Tpe(Mim), that were mostly designed for accuracy perform well if the metric favors reduction and compression of the feature vector. This is the case for metrics, such as feature set size, privacy, and safety. However, if the metric requires the selection strategy to prune specific features that are unrelated to accuracy, such as biased features in the case of EO, accuracy-optimized rankings fail for high thresholds, strategies that consider a larger search space, such as Tpe(Nr) and SA(Nr), perform better.

Table 8: Combinations maximizing coverage and fastest.

top-k	Objective: Coverage		Objective: Fastest	
	Combination	Achieved	Combination	Achieved
1	TPE(FCBF)	0.60 ± 0.22	SFFS(NR)	0.12 ± 0.12
2	+ SFFS(NR)	0.83 ± 0.11	+ TPE(FCBF)	0.23 ± 0.21
3	+ TPE(NR)	0.88 ± 0.08	+ ES(NR)	0.34 ± 0.22
4	+ TPE(MIM)	0.92 ± 0.06	+ SFS(NR)	0.44 ± 0.28
5	+ SA(NR)	0.94 ± 0.04	+ NSGA-II(NR)	0.52 ± 0.26
6	+ TPE(χ^2)	0.96 ± 0.03	+ TPE(NR)	0.59 ± 0.23
7	+ TPE(Variance)	0.97 ± 0.03	+ SA(NR)	0.66 ± 0.22
8	+ NSGA-II(NR)	0.98 ± 0.02	+ TPE(Variance)	0.72 ± 0.19
9	+ SFS(NR)	0.99 ± 0.02	+ TPE(χ^2)	0.78 ± 0.16
10	+ TPE(Fisher)	0.99 ± 0.02	+ Original Feature Set	0.82 ± 0.13
11	+ ES(NR)	0.99 ± 0.01	+ TPE(MIM)	0.87 ± 0.11
12	+ TPE(ReliefF)	1.00 ± 0.01	+ TPE(Fisher)	0.90 ± 0.08
13	+ SBFS(NR)	1.00 ± 0.01	+ SBFS(NR)	0.93 ± 0.07
14	+ TPE(MCFS)	1.00 ± 0.00	+ SBS(NR)	0.95 ± 0.05
15			+ TPE(ReliefF)	0.97 ± 0.04
16			+ RFE(Model)	0.99 ± 0.03
17			+ TPE(MCFS)	1.00 ± 0.00

6.5 Strategy Combination Effectiveness

Instead of searching the feature subsets just with one strategy, one could run multiple strategies in parallel. The user might be interested in fast answering or high coverage. We report the combination of the top-k strategies that lead to the fastest results or the highest coverage across datasets in Table 8. The reported numbers assume embarrassingly parallel execution without interference or reuse opportunities. Therefore, this approach can be further optimized e.g., by cashing common computations [69].

The best strategy with respect to coverage is Tpe(Fcbf). Adding the sequential FS strategy Sffs(Nr) shows the highest benefit because it can satisfy more diverse and restrictive constraints. If we add the Tpe(Nr), we gain 5% in coverage on average because it covers diverse constraints and does start with a small number of features. Using 14 strategies, we yield 100% coverage. The strategies Sbs(Nr) and Rfe(Model) did not find any feature sets that cannot be covered by those 14 strategies. The best strategy with respect to fast answers is Sffs(Nr) for ML scenarios with high constraints. For easily satisfiable ML scenarios, Tpe(Fcbf) is fastest. Running both

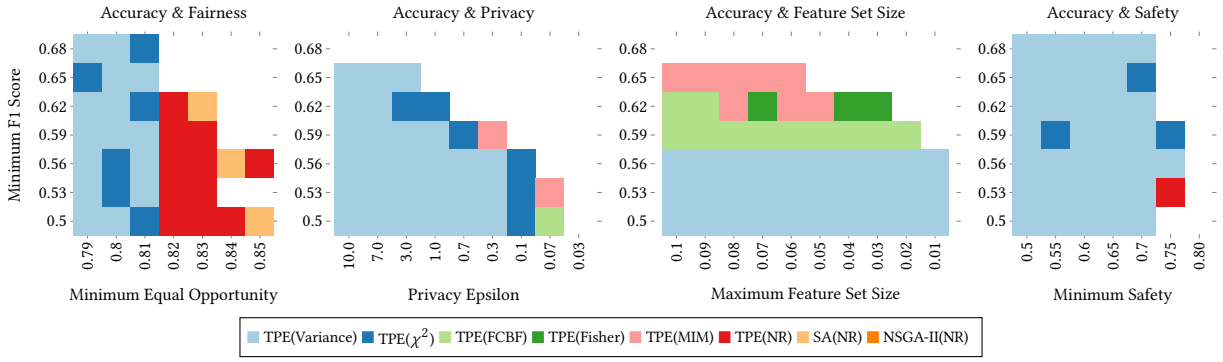

Figure 5: Fastest strategy for 4 constraint pairs on the Adult dataset.

Table 9: Meta-learning accuracy across strategies.

Strategy	Precision	Recall	F1 score
SBS(NR)	0.67 ± 0.33	0.50 ± 0.29	0.53 ± 0.28
SBFS(NR)	0.66 ± 0.30	0.52 ± 0.26	0.54 ± 0.23
RFE(Model)	0.69 ± 0.28	0.56 ± 0.26	0.57 ± 0.23
TPE(MCFS)	0.49 ± 0.36	0.41 ± 0.34	0.36 ± 0.27
TPE(ReliefF)	0.56 ± 0.30	0.58 ± 0.34	0.55 ± 0.30
TPE(Variance)	0.61 ± 0.29	0.65 ± 0.26	0.58 ± 0.23
TPE(NR)	0.63 ± 0.25	0.64 ± 0.25	0.58 ± 0.21
NSGA-II(NR)	0.63 ± 0.22	0.71 ± 0.23	0.64 ± 0.21
TPE(MIM)	0.59 ± 0.29	0.70 ± 0.35	0.62 ± 0.29
SA(NR)	0.71 ± 0.18	0.73 ± 0.15	0.70 ± 0.13
ES(NR)	0.55 ± 0.33	0.62 ± 0.37	0.56 ± 0.32
TPE(Fisher)	0.67 ± 0.28	0.66 ± 0.32	0.63 ± 0.28
TPE(χ^2)	0.71 ± 0.21	0.76 ± 0.18	0.69 ± 0.16
SFS(NR)	0.57 ± 0.33	0.67 ± 0.36	0.59 ± 0.32
SFFS(NR)	0.58 ± 0.33	0.69 ± 0.38	0.61 ± 0.33
TPE(FCBF)	0.69 ± 0.22	0.74 ± 0.26	0.68 ± 0.20

strategies in parallel results for 11% more cases the fastest result. In conclusion, parallelization is an effective approach to solve more ML scenarios or to solve ML scenarios faster. For instance, running 5 strategies in parallel leads to 94% coverage or 52% fastest answers.

6.6 Potentials of the Dfs Optimizer

Table 3 shows that our meta-learning-based Dfs optimizer improves the coverage of the best single strategy Tpe(Fcbf) by 10% on average and reduces the standard deviation by 4%. The Dfs optimizer achieves high coverage more consistently across datasets as we see in Figure 4. When Sffs(Nr) achieves poor coverage compared to other strategies, e.g. for Adult and KDD Internet Usage, our optimizer chooses the strategy that achieves consistently good coverage. Even though the Dfs optimizer is optimized for coverage, it chooses the fastest strategy in 11% on average, which is close to the fastest strategy Sffs(Nr). The optimizer learns one model for each FS strategy to predict whether it will satisfy a given ML scenario. In Table 9, we report the precision, the recall, and the F1 score for this classification task per strategy. The averaged F1 scores are fair with 70% at most. However, when each model contributes enough information, together they are accurate enough the outperform the single-best strategy and lead to much more consistent coverage.

7 CONCLUSION AND FUTURE WORK

We performed an extensive empirical study to evaluate existing FS strategies for Dfs. Based on these experiments, we investigated whether we can learn an optimizer that automatically chooses the strategy that is the most promising in satisfying the user-specified constraints. We also examined how the parallelization of these strategies can improve speed and coverage. Our experimental results led us to the following conclusions. First, Dfs is an intuitive and simple framework for enforcing constraints. The underlying FS algorithm is however highly dependent on the constraints and datasets, which motivates the use of a holistic optimizer. Our meta learning-driven Dfs optimizer improves on the coverage of the best FS algorithm by 10%. There are several promising future directions:

- **Meta learning**. We presented a meta learning-driven Dfs optimizer that can choose the best strategy for a fraction of cases. One possible direction to improve meta learning is to allow dynamic selection of strategies during runtime. One could learn an additional model that estimates after each feature evaluation whether the chosen strategy is likely to converge within the user-specified search time. If this estimate is pessimistic, we can switch to a different strategy. The new strategy could be warm-started based on the experience gained in previous runs.
- **Feature construction**. It might not be enough to only select features from the given dataset to satisfy all constraints [21, 22]. In some cases, one could additionally generate new features. Feature construction can help to uncover nonlinear relationships between the original features.
- **AutoML**. We showed that with only 5 out of 16 strategies, we can cover 94% of the Dfs-satisfiable ML scenarios. This insight allows the AutoML approaches [23, 60] to significantly decrease the search space for FS. Data science frameworks, such as SystemDS [9] could use these insights to incorporate and optimize FS components. Further, one can extend Dfs to *declarative AutoML* where we not only select features but also models and hyperparameters to satisfy user-specified constraints.

Acknowledgments. This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

REFERENCES

[1] David W Aha and Richard L Bankert. 1996. A comparative evaluation of sequential feature selection algorithms. In *Learning from data*. 199–206.
[2] Michael R Anderson, Dolan Antenucci, Victor Bittorf, Matthew Burgess, Michael J Cafarella, Arun Kumar, Feng Niu, Yongjoo Park, Christopher Ré, and Ce Zhang. 2013. Brainwash: A Data System for Feature Engineering.. In *CIDR*.
[3] Michael R Anderson and Michael Cafarella. 2016. Input selection for fast feature engineering. In *ICDE*. 577–588.

[4] Michael R. Anderson, Michael Cafarella, Yixing Jiang, Guan Wang, and Bochun Zhang. 2014. An Integrated Development Environment for Faster Feature Engineering. *PVLDB* 7, 13 (2014), 1657–1660.

[5] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *ICML*. 233–242.

[6] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜. In *FAccT*.

[7] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *NeurIPS*. 2546–2554.

[8] Julian Blank and Kalyanmoy Deb. 2020. Pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[9] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*.

[10] Matthias Boehm, Michael W. Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Arvind C. Surve, and Shirish Tatikonda. 2016. SystemML: Declarative Machine Learning on Spark. *PVLDB* 9, 13 (2016), 1425–1436.

[11] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[12] Deng Cai, Chiyuan Zhang, and Xiaofei He. 2010. Unsupervised feature selection for multi-cluster data. In *SIGKDD*. 333–342.

[13] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. 2011. Differentially private empirical risk minimization. *JMLR* 12, Mar (2011), 1069–1109.

[14] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. 2019. Robustness verification of tree-based models. In *NeurIPS*. 12317–12328.

[15] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. 2020. Hopskipjumpattack: A query-efficient decision-based attack. In *SP*. 1277–1294.

[16] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Ziawasch Abedjan, Tilmann Rabl, and Volker Markl. 2020. Optimizing Machine Learning Workloads in Collaborative Environments. In *SIGMOD*.

[17] Justin Doak. 1992. An evaluation of feature selection methods and their application to computer security. *Technical Report CSE-92-18* (1992).

[18] Michele Donini, Luca Oneto, Shai Ben-David, John S Shawe-Taylor, and Massimiliano Pontil. 2018. Empirical risk minimization under fairness constraints. In *NeurIPS*. 2791–2801.

[19] Richard O Duda, Peter E Hart, and David G Stork. 2012. *Pattern classification*. John Wiley & Sons.

[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *TCC*. 265–284.

[21] Mahdi Esmailoghli and Ziawasch Abedjan. 2020. CAFE: Constraint-Aware Feature Extraction from Large Databases. In *CIDR*.

[22] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In *EDBT*. 331–336.

[23] Matthias Feurer et al. 2015. Efficient and robust automated machine learning. In *NeurIPS*. 2962–2970.

[24] Sam Fletcher and Md Zahidul Islam. 2017. Differentially private random decision forests using smooth sensitivity. *Expert Systems with Applications* 78 (2017), 16–31.

[25] Johannes Fürnkranz and Johann Petrak. 2001. An evaluation of landmarking variants. In *ECML/PKDD*. 57–68.

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR*.

[27] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *JMLR* 3 (2003), 1157–1182.

[28] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene selection for cancer classification using support vector machines. *Machine learning* 46, 1-3 (2002), 389–422.

[29] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of opportunity in supervised learning. In *NeurIPS*. 3315–3323.

[30] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *DSAA*. 1–10.

[31] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Saravanan Thirumuruganathan, Sanjay Chawla, and Divy Agrawal. 2017. A cost-based optimizer for gradient descent optimization. In *SIGMOD*. 977–992.

[32] Ron Kohavi, George H John, et al. 1997. Wrappers for feature subset selection. *AI* 97, 1-2 (1997), 273–324.

[33] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*. 1717–1722.

[34] Arun Kumar, Jeffrey Naughton, Jignesh M Patel, and Xiaojin Zhu. 2016. To join or not to join? thinking twice about joins before feature selection. In *SIGMOD*. 19–34.

[35] Jeff Larson, Surya Mattu, Lauren Kirchner, and Julia Angwin. 2016. How we analyzed the COMPAS recidivism algorithm. *ProPublica (5 2016)* 9, 1 (2016).

[36] David D Lewis. 1992. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 212–217.

[37] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. 2017. Feature selection: A data perspective. *CSUR* 50, 6 (2017), 1–45.

[38] Huan Liu and Rudy Setiono. 1995. Chi2: Feature selection and discretization of numeric attributes. In *ICTAI*. 388–391.

[39] Frank McSherry and Ilya Mironov. 2009. Differentially private recommender system. *KDD* (2009).

[40] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics* 21, 6 (1953), 1087–1092.

[41] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. 2002. Feature selection algorithms: A survey and experimental evaluation. In *ICDM*. 306–313.

[42] John Moody. 1989. Fast learning in multi-resolution hierarchies. In *NeurIPS*. 29–39.

[43] Arvind Narayanan. 2018. 21 fairness definitions and their politics. In *FAccT*.

[44] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. 2017. Learning Feature Engineering for Classification.. In *IJCAI*. 2529–2535.

[45] Felix Neutatz. 2021. DFS. https://github.com/BigDaMa/DFS.

[46] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NeurIPS*. 849–856.

[47] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. 2018. Adversarial Robustness Toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069* (2018).

[48] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. 2018. Adversarial Robustness Toolbox v1.1.1. *CoRR* 1807.01069 (2018). https://arxiv.org/pdf/1807.01069

[49] Feiping Nie, Heng Huang, Xiao Cai, and Chris H Ding. 2010. Efficient and robust feature selection via joint l2, 1-norms minimization. In *NeurIPS*. 1813–1821.

[50] Karl Pearson. 1901. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12 (2011), 2825–2830.

[52] Valerio Perrone, Michele Donini, Krishnaram Kenthapadi, and Cédric Archambeau. 2020. Fair Bayesian Optimization. *arXiv preprint arXiv:2006.05109* (2020).

[53] Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Vaughan, and Hanna Wallach. 2018. Manipulating and measuring model interpretability. *arXiv preprint arXiv:1802.07810* (2018).

[54] Pavel Pudil, Jana Novovičová, and Josef Kittler. 1994. Floating search methods in feature selection. *Pattern recognition letters* 15, 11 (1994), 1119–1125.

[55] Marko Robnik-Šikonja and Igor Kononenko. 2003. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine learning* 53, 1-2 (2003), 23–69.

[56] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional fairness: Causal database repair for algorithmic fairness. In *SIGMOD*. 793–810.

[57] Sebastian Schelter, Yuxuan He, Jatin Khilnani, and Julia Stoyanovich. 2020. FairPrep: Promoting Data to a First-Class Citizen in Studies on Fairness-Enhancing Interventions. In *EDBT*. 395–398.

[58] Andrew D. Selbst. 2017. Disparate Impact in Big Data Policing. *Georgia law review* 52 (2017), 3373.

[59] Andreas Seltenreich, Bo Tang, and Sjoerd Mullender. 2021. *SQLsmith*. https://github.com/anse1/sqlsmith

[60] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing data science through interactive curation of ml pipelines. In *SIGMOD*. 1171–1188.

[61] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2017. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. In *ICDE*. 535–546.

[62] Till Speicher, Hoda Heidari, Nina Grgic-Hlaca, Krishna P. Gummadi, Adish Singla, Adrian Weller, and Muhammad Bilal Zafar. 2018. A Unified Approach to Quantifying Algorithmic Unfairness: Measuring Individual &Group Unfairness via Inequality Indices. In *SIGKDD*. 2239–2248.

[63] Julia Stoyanovich, Bill Howe, and H. V. Jagadish. 2020. Responsible Data Management. *PVLDB* 13, 12 (2020), 3474–3488.

[64] TPC. 2021. *TPC-H*. http://www.tpc.org/tpch/

[65] Ryan J. Urbanowicz, Randal S. Olson, Peter Schmitt, Melissa Meeker, and Jason H. Moore. 2018. Benchmarking relief-based feature selection methods for

bioinformatics data mining. *J. Biomed. Informatics* 85 (2018), 168–188.
[66] Jaideep Vaidya, Basit Shafiq, Anirban Basu, and Yuan Hong. 2013. Differentially private naive bayes classification. In *WI*, Vol. 1. 571–576.
[67] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60.
[68] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. 2018. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. In *ICLR*.
[69] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya Parameswaran. 2018. Helix: Holistic optimization for accelerating iterative machine learning. *PVLDB* 12, 4 (2018), 446–460.
[70] Bing Xue, Mengjie Zhang, and Will N Browne. 2012. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE transactions on cybernetics* 43, 6 (2012), 1656–1671.
[71] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *TEVC* 20, 4 (2015), 606–626.
[72] Anatoly Yakovlev, Hesam Fathi Moghadam, Ali Moharrer, Jingxiao Cai, Nikan Chavoshi, Venkatanathan Varadarajan, Sandeep R Agrawal, Sam Idicula, Tomas Karnagel, Sanjay Jinturkar, et al. 2020. Oracle AutoML: a fast and predictive AutoML pipeline. *PVLDB* 13, 12 (2020), 3166–3180.
[73] Lei Yu and Huan Liu. 2003. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*. 856–863.
[74] Ce Zhang, Arun Kumar, and Christopher Ré. 2016. Materialization optimizations for feature selection workloads. *TODS* 41, 1 (2016), 1–32.
[75] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. 2020. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* (2020).