Effective pruning for top-k feature search on the basis of SHAP values

Lisa Chabrier, Anton Crombach, Sergio Peignier, Christophe Rigotti

Effective pruning for top-k feature search on the basis of SHAP values

Lisa Chabrier[1, 2], Anton Crombach[1, 2], Sergio Peignier[3], and Christophe Rigotti[1, 2]

[1]Inria, Lyon Centre, France.
[2]INSA Lyon, CNRS, UCBL, LIRIS, UMR5205, F-69621 Villeurbanne, France.
[3]BF2i, UMR 0203 INRAE, INSA Lyon, France.

Financial support: This work was partially funded by PEPR Santé Numérique, project 22-PESN-0002; Fondation ARC under grant ARCPJA22020060002212; Institut National du Cancer under grant PLBIO22-071; the National Institutes of Health (NIH) award R01DC020478; BQR INSA Lyon 2023 Neurinfo; and ANR project C2R-IA ANR-22-CE56-0005.

Abstract

With the increasingly pervasive use of advanced machine learning models comes the need to explain their predictions. The SHAP framework, based on Shapley values, provides explanations to highlight which features could be important for a given prediction. However, its use is hampered by its computational cost, especially in its model-agnostic formulation. Model-specific algorithms offer a restricted solution to this problem, whereas alternative approximation strategies can maintain the model-agnostic property. Here we propose TopShap as an agnostic algorithm that searches the k most important features by interleaving pruning of candidates and refinement of the approximate SHAP values. TopShap is built on three insights: (i) it performs an iterative approximation taking advantage of a previously developed sampling strategy, (ii) it uses confidence interval bounds around approximate SHAP values to determine on-the-fly which features can no longer be part of the top-k, (iii) it stops when these interval bounds are stable. Evaluating TopShap on publicly available datasets shows it performs an effective pruning of the feature search space and leads to an important reduction of the execution cost when compared to the other agnostic approaches. We also apply TopShap to a use case in biology and show that top-k search is meaningful in this context.

Keywords: Explainable artificial intelligence, local feature attribution, model-agnostic method, Shapley value, top-k selection.

1 Introduction

Machine learning models have become more and more sophisticated over the last decade and the need to explain their predictions has been increasing similarly [1]. Indeed, understanding how machine learning models make their predictions can help to decipher and counter bias in training data, and enables domain-experts to judge model output, which in turn builds trust in model decisions. One way to explain the predictions of a model, is to provide importance values for the features used in the input of the model. This can be done at the global level, encompassing all predictions, or at the local level, where feature importances are assessed for a single prediction made for a given input instance.

Among local feature importance approaches, so-called SHAP values [2] have been shown to give meaningful insight into machine learning predictions (e.g., [3, 4]). SHAP values are based on a game theory approach developed by Lloyd Shapley [5], addressing the distribution of gain between players in cooperative multiplayer games. In the context of explaining predictions made by machine learning models, the goal is to quantify the influence of each feature on the output of the model. SHAP values are the resulting framework, using the desirable properties of Shapley values [5] and the requirement of additive feature attributions as found in a larger group of explainability methods [6, 7].

One of the main difficulties of using SHAP values, is their computational cost, because they require to take into account the contribution of each subset of features and they need to obtain the expectation of the model output when only such a subset is known. An important research direction is the reduction of these costs, while being independent of the type of models, i.e., being model-agnostic, in order to be of benefit to any current (or emerging) types of models. In existing works (e.g., [8,9]), these gains are made by using sampling strategies to compute approximations of feature importance measures. It should be noted, that for some types of models, other improvements can be obtained by taking advantage of the model structure itself, though this implies losing the property of being model-agnostic [3].

Knowing the importance of all features can deliver meaningful insight, yet an explanation of model behavior that highlights which features contribute the most to a prediction, could be sufficient for the user in order to develop an understanding of that prediction. In this paper,

we preserve the model-agnostic perspective, and we propose to further reduce the cost of the sampling strategies by focusing on the k most important features according to their prediction attribution.

The main contribution of our work, is to show that such a top-k selection and corresponding SHAP value computation can benefit from an effective pruning of the search space, avoiding the need to compute the precise importance of most features not in the top-k. As the computational cost of SHAP values is exponential in the number of features, such a reduction of the search space is particularly interesting for datasets with a large number of features.

To perform such a pruning, an algorithm is proposed based on an iterative sampling strategy that interleaves bound improvement of SHAP value estimates and pruning of the set of features that are candidates to lie in the top-k. Experiments on various datasets and representative types of models exhibit a fast reduction of the number of candidates, thus avoiding the correspondingly large number of computationally expensive sampling operations. When compared to current existing alternative approaches, this leads to an important reduction of the execution cost. In addition, we show in a use case from biology, that such a top-k selection provides meaningful insight in the context of gene expression data analysis.

The rest of this paper is organized as follows. The next section recalls the necessary preliminaries. Section 3 presents related works. The algorithm to compute the top-k features is described in Section 4. Section 5 reports the experiments and we conclude with a summary in Section 6. Complementary experiment results are reported in the Supplementary Materials.

2 Preliminaries

In the following, the term *SHAP value* designates the importance of a feature in a prediction as defined by both Strumbelj *et al.* [7] and the unified framework of SHapley Additive exPlanations [2].

Let f be an already trained regression model, that given an instance x described by a set of features F, can be used to compute $f(x) \in \mathbb{R}$, the prediction associated to x. For each feature $i \in F$, the SHAP value $\phi_i(f, x)$ assesses the contribution of i in the prediction $f(x)$.

Let $f_S(x)$, with $S \subseteq F$, denote the output of f when only the values of the features in S are known for x. For instance, $f_\emptyset(x)$ is the default prediction of f when no feature is known.

The main property of the SHAP values is the local accuracy, which guarantees that their addition to $f_\emptyset(x)$ sums to the prediction for input x:

$$f(x) = f_\emptyset(x) + \sum_{i \in F} \phi_i(f, x). \tag{1}$$

Note that in this additive explanation framework, the contribution of a feature can be positive or negative, and that features with high importance are the ones with large absolute $\phi_i(f, x)$ values.

Following the formal framework of [5], it has been shown that, under reasonable fairness axioms (e.g., symmetry, consistency), there exists only one function ϕ_i and that it can be defined as [2, 7]:

$$\phi_i(f, x) = \sum_{S \subseteq F \setminus \{i\}} w(S)\big(f_{S\cup\{i\}}(x) - f_S(x)\big), \tag{2}$$

where w is a weight function depending on the size of S and F:

$$w(S) = \frac{|S|!\,(|F| - |S| - 1)!}{|F|!}. \tag{3}$$

This follows the cooperative game theory results of [5], which proposes a solution to the problem of allocating benefits to players under fairness axioms. The analogy is that the set of features F is replaced by a set of players, and the function $f_S(x)$ is replaced by a function $\nu(S)$ for $S \subseteq F$ that gives the value of the coalition of players S in the game. In this original context, the allocation of a player $i \in F$ was defined as:

$$\mathcal{A}_i(\nu) = \sum_{S \subseteq F \setminus \{i\}} w(S)\big(\nu(S \cup \{i\}) - \nu(S)\big). \tag{4}$$

Having recalled the definitions necessary to introduce the TopShap algorithm, we note that, for the sake of clarity, the above preliminaries have been restricted to regression tasks. However, SHAP values can be easily reformulated for classification tasks. For instance, in the case of a binary classification with classes $\mathcal{C}_0$ and $\mathcal{C}_1$, if $f(x)$ outputs the probability to belong to class $\mathcal{C}_1$, with $1 - f(x)$ being the probability that x lies in $\mathcal{C}_0$, then the framework can be applied directly.

3 Related Work

Computing $\phi_i(f, x)$ or $\mathcal{A}_i(\nu)$, according to (2) and (4), suggests a summation over the subsets of F. The cost can be reduced for particular classes of cooperative games (e.g., [10]), but, in general, computing the Shapley values as defined by (4) requires an exponential number of operations [11]. There exist two other equivalent closed-form expressions of the Shapley values. The first relies on the fact that these values can be shown to be the optimal solutions of a Weighted Least Squares (WLS) problem [12], where, however, the exact solutions still require the evaluation of an exponential number of terms. The second one is based on rewriting (4) as an average over permutations of F instead of a weighted sum over subsets of F. It gave rise to the method proposed by [13] to estimate the Shapley values for real world problems (when $|F|$ is large) using a uniform random sampling of the permutations.

In the context of local feature attribution of a prediction, the SHAP values suffer from the same exponential computation cost as Shapley values. It is even worse, because $\phi_i(f, x)$ in (2) requires also to determine $f_S(x)$,

that is the expected output of the model when only a subset S of the features of x are known. Reducing the cost of computing SHAP values has received ample attention in two main research directions. The first one (e.g., [7], Kernel SHAP [2], [9]) aims to be broadly applicable and therefore takes a model-agnostic perspective, where SHAP computations are kept independent from the kind of models. The second one, on the contrary, proposes non-agnostic approaches (e.g., Tree SHAP [3], Linear SHAP [2]), that are dedicated to one kind of model and speed-up the computation by taking advantage of the model structure itself.

Here, we pursue the general model-agnostic option and propose a top-k approach that does not rely on model-specific considerations. In the community, the search for methods to approximate SHAP values while preserving the model-agnostic property have led to two main fruitful families of methods, similar to those used for Shapley values: sampling the permutations of F or solving an equivalent WLS problem.

In the first family, among the approaches based on sampling permutations (e.g., [9]) the one of [14] is of particular interest because it incorporates the approximation of $f_S(x)$ in the sampling process. Indeed, computing $f_S(x)$ is a problem in itself, and most kinds of models cannot output it directly. The algorithm presented in [14] solved this problem by using a joint sampling of permutations and of values in the domain of x to estimate $f_S(x)$.

In the second family, the WLS approach led to the method of reference called Kernel SHAP [2], shown to be more efficient than the sampling of [14] mentioned above. As detailed in [15], the approximation made by Kernel SHAP is also based on sampling, but it is a sampling over the subsets of F, not over the permutations of F. Sampling was used by this algorithm to reduce the size of the set of equations involved in solving the WLS problem to obtain SHAP values.

Beyond the reduction of computational cost, other studies tried to put a strict limit on global resource consumption. Such an algorithm was for instance proposed by [14], aiming to obtain the best approximations for all $\phi_i(f, x)$ using no more than a given total number of observations drawn to build all samples, i.e., satisfying a sampling budget constraint. For cooperative games, such a constraint has also been used by [16] to limit the resources allocated to the computation of the $\mathcal{A}_i(\nu)$ of the different players. In addition, the authors proposed explicitly to output only the top-k players, as follows. Given a maximum number of drawing operations allowed by the total budget, the algorithm refines the approximations of the $\mathcal{A}_i(\nu)$ as much as possible. Then, in a post-processing step, the top-k are selected. Computing the top-k $\mathcal{A}_i(\nu)$ has been used in [17] to identify influential nodes in social networks, but the top-k selection was again simply made as a post-processing step.

Our contribution is to show that, when computing the top-k features and their corresponding $\phi_i(f, x)$, an effective pruning can be interleaved within the permutation sampling process. The experiments reported in Section 5 show an important reduction of the search space during the sampling of the permutations, in comparison to a selection of the top-k based on post-processing. In addition, this pruning is compared to Kernel SHAP, the algorithm of reference for model-agnostic SHAP value computation, that has been reported to have better performance than classical sampling of permutations. Our experiments show that this is no longer the case when pruning steps are interleaved within the permutation sampling, and that the gain is strong when k is small relatively to the total number of features.

4 TopShap

In this section, we present TopShap, a model-agnostic algorithm to compute the top-k features according to their SHAP values. Since the contribution of a feature to a prediction, as captured by its SHAP value, can be positive or negative, features need to be considered by their absolute SHAP values. In addition, defining the top-k requires to handle ties, especially because the fairness axioms of the original Shapley values impose ties for equivalent contributors [5]. Thus, we define a top-k feature as having no more than $k-1$ other features with a strictly greater absolute SHAP value.

Definition 1 (top-k features)**.** *A feature i is a top-k feature of a set of features F, for a model f and an instance x if and only if*

$$\left|\{j \in F \mid |\phi_j(f,x)| > |\phi_i(f,x)|\}\right| < k. \tag{5}$$

To reduce computational cost, searching for the top-k is a viable strategy if it can be done without a naive determination of all SHAP values. This means that, during the computation, a bounding and pruning mechanism is needed to avoid estimating SHAP values for features that can no longer be in the top-k.

Deriving bounds for player allocations and SHAP value using sampling has been studied by [13] and [7]. Noting that the allocation $\mathcal{A}_i(\nu)$ in (4) can be equivalently obtained as an average over the permutations of the elements of F (detailed later), the authors of [13] showed that $\mathcal{A}_i(\nu)$ can be estimated as the mean of a sequence of *i.i.d.* random variables, providing an unbiased and consistent estimator. Applying the central limit theorem, they derived a confidence interval for $\mathcal{A}_i(\nu)$. This approach was extended by [7] to estimate $\phi_i(f, x)$ for a model f, while at the same time sampling over the domain of x to estimate the terms $f_S(x)$ and $f_{S\cup\{i\}}$ in (2).

The aforementioned bounding scheme is adopted in TopShap to reduce incrementally the number of candidate features on the basis of improved confidence interval bounds through additional sampling (i.e., increasing the length of the sequence of random variables used). Starting from an initial set of candidate features CF that is equal to the whole feature set, the core algorithm repeats the following three steps: (1) perform sampling for features in CF; (2) compute new bounds of their SHAP values; (3) prune useless candidates from CF.

At first glance, the stopping criterion could be the presence of at most k different $\phi_i(f,x)$ values among the features in CF, with CF having possibly a size greater than k because of ties. Unfortunately, in the presence of ties, such a condition is unlikely to be reached in a reasonable number of iterations. Indeed, the computed $\phi_i(f,x)$ values are approximations and two features with the same true SHAP value can still have different intermediate estimates $\phi_i(f,x)$ until their numerical precision reaches machine precision.

To avoid this problem, the stopping criterion in TopShap is based on the stability of the confidence interval bounds. This means that the algorithm terminates when all remaining candidates have only minor bound variations in the recent past iterations.

The corresponding algorithm, TopShap, is presented in Algorithm 1 and detailed hereafter.

4.1 Main loop

The following symbols are used as global constants. N is the total number of features. F is the set of all feature identifiers, each feature being represented by an integer identifier in $\{1,\ldots,N\}$. D is the set of instances used to train the model. The symbols f and x denote the model and instance for which the SHAP values are computed. The explicit parameters are: k the requested number of top features; $warmUp$ the size of the sample used to approximate each SHAP value in the initialization stage; and the confidence level γ used to determine the confidence intervals.

Algorithm 1 starts by initializing V, up and low. V is an array where element $V[i]$ will contain the list of values to be used to estimate $\phi_i(f,x)$. The variables up and low have the same structure as V, with $up[i]$ (resp. $low[i]$) containing the list of upper bounds (resp. lower bounds) of the confidence interval for $\phi_i(f,x)$. These bounds are computed and stored at each iteration and will be used to assess the stability.

The initial set of candidate features CF is the whole set of features F (line 3) and a stage of $warmUp$ iterations is performed to compute initial estimates. A first pruning of the candidates is then applied (line 7). Next, the algorithm iteratively computes new estimations for all (remaining) candidates and attempts to prune them further, until the confidence interval bounds of all remaining candidates are stable. Algorithm 1 returns the estimates of $\phi_i(f,x)$ for the features i in CF, together with their corresponding confidence intervals.

4.2 Estimation and bounding

We adopt the estimation framework of [14], defined as follows. The population P of the sampling process is the set of all pairs $(\mathcal{O},w)$, where $\mathcal{O}$ is a permutation of the set of features F, and w is an instance. Let $Pre^i(\mathcal{O})$ be the set of features preceding feature i in permutation $\mathcal{O}$. For a pair $(\mathcal{O},w)$, and a feature i, we consider the characteristic $\delta_i(\mathcal{O},w)$ that represents the marginal contribution of feature i to the value of $f(x)$ when i completes the features in $Pre^i(\mathcal{O})$, and the values of the other features being set to those of the random instance w.

This characteristic $\delta_i(\mathcal{O},w)$ is computed as follows. Let z and z' be two instances having feature values $z_1,\ldots,z_N$ and $z'_1,\ldots,z'_N$ set by merging x and w in the following way: z_j is set to x_j if $j \in Pre^i(\mathcal{O})$, and to w_j otherwise; z'_j is set to x_j if $j=i$, and to z_j otherwise. Then, $\delta_i(\mathcal{O},w)$ is given by $\delta_i(\mathcal{O},w) = f(z') - f(z)$.

Let M be a sample of size m of pairs $(\mathcal{O},w)$ of the population P. Then, $\hat{\phi}_i(f,x)$, the estimate of $\phi_i(f,x)$, is defined as the mean of $\delta_i(\mathcal{O},w)$ over M. This estimator was shown to be unbiased and consistent [14]. Furthermore, applying the central limit theorem, it was derived that $\hat{\phi}_i(f,x) - \phi_i(f,x)$ is approximately normally distributed with mean 0 and variance $\frac{\sigma^2}{m}$, where σ^2 is the variance of $\delta_i(\mathcal{O},w)$ [14]. This allows a straightforward computation of bounds of a confidence interval for a confidence level γ. For a set $V[i]$ of the values of $\delta_i(\mathcal{O},w)$ over M, we note $\text{BOUND}(V[i],\gamma)$ the function that returns these bounds.

In TopShap, the estimation is performed iteratively calling ESTIMATE (Algorithm 2), where, for each feature i remaining in CF, a new value $\delta_i(\mathcal{O},w)$ is computed and appended to current $V[i]$ (line 5). The upper and lower bounds of the confidence interval are computed by BOUND from $V[i]$ using the unbiased sample variance as estimator for the population variance. TopShap computes these bounds for all the features in CF. So, to ensure an overall confidence level γ, we apply the extension of the Bonferroni correction to confidence intervals, as proposed by [18]. This correction is obtained by using for each intervals the confidence level $\gamma' = 1 - \frac{1-\gamma}{|CF|}$. This is done when calling BOUND (line 7).

We note that this sampling process assumes feature independence (as most other frameworks, e.g., [2]); in case of dependencies, various extensions can handle them (see [8]).

4.3 Pruning and stability

The pruning is performed in PRUNE (Algorithm 3) when there are more than k candidate features. The top-k features are defined using absolute SHAP values, whereas the confidence interval bounds in up and low are the ones corresponding to the SHAP values themselves, which can be positive or negative. Thus, first, bounds are transformed into new bounds $absUp$ and $absLow$ corresponding to absolute SHAP values. For a given feature, $absUp$ is simply the maximum of the absolute values of the two bounds (line 8). For $absMin$, if the signs of the two bounds are the same, then $absMin$ is the minimum of the absolute values of the bounds (line 11). If they have different signs, then it is zero. The pruning itself is done as follows. First, the k^{th} highest element among the new lower bounds $absLow$, termed θ, is selected. Then, only the features that have an absolute upper bound $absUp$ greater or equal to θ are kept as candidates (line 18), since the others can no longer be in the set of top-k features. Note that $absUp$ is stored as an array to allow constant time direct access. Algorithm 3 finishes by returning the new set of candidate features, and its correctness is shown

below.

Theorem 1. *The selection made by Algorithm 3 is correct.*

Proof. Let us suppose that CF contains at least the top-k features. If they are no more than k candidate features (i.e., $|CF| \leq k$), all features are selected and correctness is trivial. Thus, in the following, let us also suppose that $|CF| > k$.

The completeness of the selection comes from the safety of the pruning and can be shown by contradiction. Suppose that $i \in CF$ is in the top-k but is not returned in $newCF$. By Definition 1, i being in the top-k implies that there are strictly less than k other features having an absolute SHAP value strictly greater than the absolute SHAP value of i. So, there are strictly less than k other features having an absolute SHAP value lower bound strictly greater than the upper bound of the absolute SHAP value of i. In Algorithm 3, θ is the k^{th} lower bound (in decreasing order). Thus, i is selected as a member of $newCF$ (line 18), which contradicts the hypothesis.

The soundness of the selection is the property ensuring that a feature $i \in CF$ will not be output in $newCF$, if it is not in the top-k according to the bounds of its absolute SHAP value, $absLow$ and $absUp$. This property is straightforward, since if there are at least k other features with a lower bound of their absolute SHAP value strictly greater than the upper bound of the absolute SHAP value of i, then i is not selected in the output (line 18). □

In Algorithm 1, the pruning is interleaved with the computation of new estimations up to reaching stable bounds. This stability is checked by ALL - CF - STABLE which returns *true* if and only if, for all remaining candidate features, the variation of the size of the confidence interval was strictly less than 0.1% between two consecutive estimations over the previous 100 iterations.

Algorithm 1 TopShap

1: **Input** k, γ, $warmUp$
2: Initialize V, up and low as arrays of size N, each containing N empty lists.
3: $CF \leftarrow F$
4: **for** initialEstim $\leftarrow 1$ to $warmUp$ **do**
5: $\quad V, up, low \leftarrow$ ESTIMATE(CF, V, up, low, γ)
6: **end for**
7: $CF \leftarrow$ PRUNE(CF, up, low, k)
8: **while** *not*(ALL - CF - STABLE(CF, up, low)) **do**
9: $\quad V, up, low \leftarrow$ ESTIMATE(CF, V, up, low, γ)
10: $\quad CF \leftarrow$ PRUNE(CF, up, low, k)
11: **end while**
12: **Output** Set CF, estimates $mean(V[i])$ for the features i in CF, and corresponding confidence intervals $[last(low[i]), last(up[i])]$.

Algorithm 2 ESTIMATE

1: **Input** CF, V, up, low, γ
2: $\mathcal{O} \leftarrow$ random permutation of F
3: $w \leftarrow$ random instance drawn from D
4: **for** i in CF **do**
5: $\quad$ append $\delta_i(\mathcal{O}, w)$ to $V[i]$
6: $\quad \gamma' \leftarrow 1 - \frac{1-\gamma}{|CF|}$
7: $\quad u, \ell \leftarrow$ BOUND$(V[i], \gamma')$
8: $\quad$ append u to $up[i]$
9: $\quad$ append ℓ to $low[i]$
10: **end for**
11: **Output** V, up, low

Algorithm 3 PRUNE

1: **Input** CF, up, low, k
2: **if** $|CF| \leq k$ **then**
3: $\quad newCF \leftarrow CF$
4: **else**
5: $\quad$ Initialize $absUp$ as an array of size N.
6: $\quad absLow \leftarrow$ empty list
7: $\quad$ **for** i in CF **do**
8: $\quad\quad absUp[i] \leftarrow max(|last(up[i])|, |last(low[i])|)$
9: $\quad\quad$ **if** $sign(last(up[i])) = sign(last(low[i]))$ **then**
10: $\quad\quad\quad$ append $min(|last(up[i])|, |last(low[i])|)$
11: $\quad\quad\quad\quad$ to $absLow$
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ append 0 to $absLow$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ sort $absLow$ in decreasing order
17: $\quad \theta \leftarrow absLow[k]$
18: $\quad newCF \leftarrow \{i \in CF \mid absUp[i] \geq \theta\}$
19: **end if**
20: **Output** $newCF$

4.4 Time complexity

An iteration (Algorithm 1, line 8) starts by a call to ALL - CF - STABLE performed in $O(|CF|)$. Then, in ESTIMATE, a permutation of F is drawn in $O(|F|)$, followed by the computation of $\delta_i(\mathcal{O}, w)$ needing to merge x and w which can be done once before the *for* loop (line 4) in $O(|F|)$. It also requires to evaluate the model f on two instances, with a cost $modelEvalCost$ that depends on the type of model. The evaluation of BOUND can be performed in $O(1)$ using an incremental update of the mean and of the variance [19]. Thus, the cost of ESTIMATE is in $O(|F| + |CF| \times modelEvalCost)$.

In PRUNE, setting/inserting elements in $absUp, absLow$ and $newCF$ (loop line 7 and line 18) is in $O(|CF|)$. Finding the k^{th} highest element (lines 16-17) of $absLow$ (of size $|CF|$) can be done in $O(|CF| \log k)$ using a min-heap.

Thus, the overall cost of an iteration in TopShap is in $O(|F| + |CF|(modelEvalCost + \log k))$. This is to be compared to the complexity of a top-k search based on sampling but without pruning, where the top-k selection would be performed in a post-processing step. In the lat-

Dataset	# Features	# Instances
Concrete	8	1031
Wine Quality	11	1600
Appliances Energy	25	19735
PBMC	127	2638
Mercedes	375	4209
CT location	384	53500
Synthetic	400	5000

Table I: Number of numerical features and instances of the datasets.

ter case, the time complexity of an iteration would be $O(|F| \times modelEvalCost)$, requiring more model evaluations.

5 Experiments

In this section, experiments are presented, reporting the performance of TopShap when computing the top-k features according to their SHAP values. The goal of the experiments is to assess the pruning when compared to classical sampling and to Kernel SHAP. A Python implementation of TopShap, as well as code to reproduce the experiments, are publicly available as a git repository at `https://gitlab.inria.fr/topshap/topshap_and_experiments`.

5.1 Datasets

TopShap's pruning capabilities have been assessed on six real-world datasets and a synthetic one. These datasets exhibited different number of instances and features, among which only the numerical ones were selected. The final corresponding sizes are reported in Table I. Most of these datasets, namely *Concrete* [20], *Wine Quality* [21], *Appliances Energy* [22] and *CT location* [23], came from the UCI Machine Learning Repository [24]. The *Mercedes* dataset was provided by Mercedes-Benz Greener Manufacturing and was accessible via Kaggle [25]. We used the *make_regression* function from scikit-learn [26] to generate the *Synthetic* dataset. This dataset contained uncorrelated random features following a normal distribution with zero mean and unitary standard deviation. The regression target variable was a random linear combination of all input features. Finally, we relied on the 10x Genomics Peripheral Blood Mononuclear Cells (*PBMC*) dataset [27] to explore the potential of the algorithm in a bioinformatics-oriented use case.

5.2 Model types

Since TopShap is a model-agnostic approach, we decided to test it with different machine-learning approaches for regression. The experiments presented in this article include four models: *Regression Forest* (*RF*), *Multi-Layer Perceptron* (*MLP*), *RBF-kernel Support Vector Regression* (*SVR*) and *Stacked Generalization* (*STK*).

For all models, scikit-learn (v1.2.2) [26] was used. For the first three models (*RF*, *MLP*, *SVR*), the hyperparameters were optimized by a grid search over the usual hyperparameter ranges for these models (details can be found in the git repository). *STK* consisted of a stacking of the other three optimized models, combined using a gradient boosting regressor with hyperparameters set to their default values.

5.3 Experimental setup

For each dataset, models were trained on a randomly chosen 70% of the instances, and the remaining 30% of the dataset was used as a test set. Then, TopShap was executed on 100 randomly chosen test instances, to search their top-k features and estimate their corresponding SHAP values. The experiments presented in the main text have been run with TopShap parameters set to k=5, confidence γ=0.95 and $warmUp$=100. For datasets with enough features for a meaningful selection, experiments were run also with k=15. To demonstrate robustness of our results, additional experiments with k=10 and γ=0.99 are provided in Supplementary Materials.

5.4 Comparison with classical sampling

The pruning proposed in TopShap is interleaved with the permutation sampling process. However, a classical sampling of permutations could be used to compute all SHAP values and then, when stability is reached, the selection of the top-k could be performed in a post-processing step. The interest of using TopShap, in comparison to such a post-processing strategy, is studied in this section.

In order to illustrate the global behavior of TopShap, we represent in Fig. 1a the evolution of the SHAP values for a *Regression Forest* model and a single instance picked from the *Wine Quality* dataset. For each of the 11 features, the absolute SHAP values, surrounded by their confidence intervals, are given along the iterations. In this example, the width of the confidence intervals shrank quickly, which enabled 6 features to be pruned from the candidates, and only 5 features were left before reaching the first 1000 iterations. The stability criterion was reached at iteration 4234 (Fig. 1a, vertical line), where TopShap stopped.

Fig. 1b and 1c illustrate the pruning performance of TopShap, for the same regression model, on 100 randomly selected test instances from the *Wine Quality* dataset. The median and average number of candidate features along the iterations are depicted in Fig. 1b. The green area demarcates the distribution of the number of candidates between the maximal and minimal values, and the left (resp. right) vertical line denotes the iteration when the first (resp. last) instance reached the stability criterion. This implies that TopShap stopped between these two lines for the 100 instances.

The behavior for one instance, as shown in Fig. 1a, is also observed over the 100 instances. Both the average and median number of candidate features decreased (from 11 to 6 features) quickly during the first 2,000 iterations (Fig. 1b). The pruning was effective and the mean and median number of candidates tended to plateau in the subsequent iterations. The number of iterations necessary to reach stability, i.e., the iteration at which TopShap stopped, was different for each instance. The histogram of the number of iterations needed for each of the 100 instances is reported in Fig. 1c. In this histogram, each bin is split according to the number of candidates remaining when TopShap stopped. If more than 5 candidates were still present, they corresponded to ties among the top-k. These ties could be features having the same SHAP values, or having overlapping confidence intervals because of the approximation framework. The precise count in this example is 59 instances that reached stability with only 5 candidates remaining, 36 instances with 6 or 7 candidates, and 5 instances with 8 candidates.

We observed correspondingly robust behavior of TopShap for all combinations of models and datasets. A subset of combinations are presented for $k = 5$ in Fig. 2, and for datasets with more than 25 features, for $k = 15$ in Fig. 3 (see Supplementary Materials Fig. S1, S2, and S4 for the other combinations). In all cases, TopShap exhibited a pattern of feature pruning similar to what we reported above (Fig. 1) and the distributions of the number iterations needed to reach stability were consistent.

Increasing k from 5 to 15 led to an increase of the number of iterations needed to reach the stopping criterion. This can be observed when comparing Fig. 3 to the bottom panels of Fig. 2. This behavior was anticipated, since adding more features is likely to require more iterations to reach stability. In agreement with this explanation, an intermediate increase of iterations was observed for the intermediate setting of $k = 10$ (see Supplementary Materials Fig. S3).

Performing the same experiments with a higher confidence level, $\gamma = 0.99$, resulted in the presence of more ties in the top-k. This happens because a higher confidence level implies wider confidence intervals and hence more opportunities for features to overlap. Having more ties has a similar effect as choosing a larger k, and accordingly we observed a slight increase in the number of iterations needed to reach stability (Supplementary Materials Fig. S5 and S6).

In summary, across a variety of data sets and for a range of parameter settings our results demonstrated the effectiveness of TopShap in pruning candidate features to avoid 'pointless' computational effort. The gain is obvious if we compare our pruning strategy to a (hypothetical) post-processing strategy to discover the top-k features, that is to say to a permutation sampling without pruning. As we reported in Section 4.4, the computational cost of an iteration of the sampling process is driven by $|CF| \times modelEvalCost$, where CF is the set of features that are still candidates to be in the top-k. Obviously, the post-processing strategy would require at least as many iterations to reach stability as the top-k

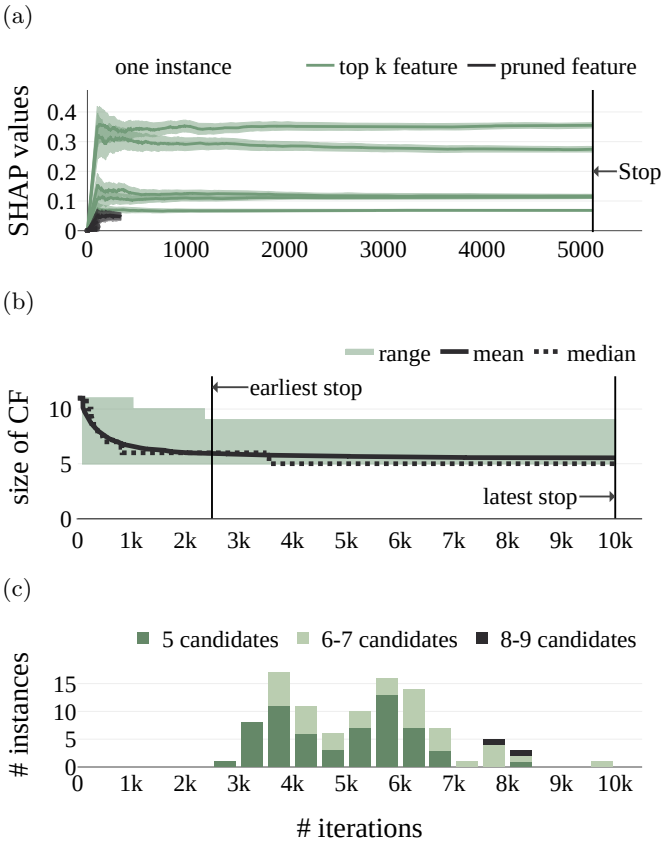

Figure 1: TopShap convergence and pruning. Dataset: *Wine Quality* (11 features); Model: *Regression Forest*. Parameters: $k = 5$, confidence $\gamma = 0.95$, warm-up iterations = 100. (a) Single instance. Pruning of features and reduction of confidence intervals. Absolute values of (approximated) SHAP values are shown. (b) 100 instances. Reduction of the number of candidates along the iterations. (c) 100 instances. Histogram of iterations for which stability is reached and TopShap stopped.

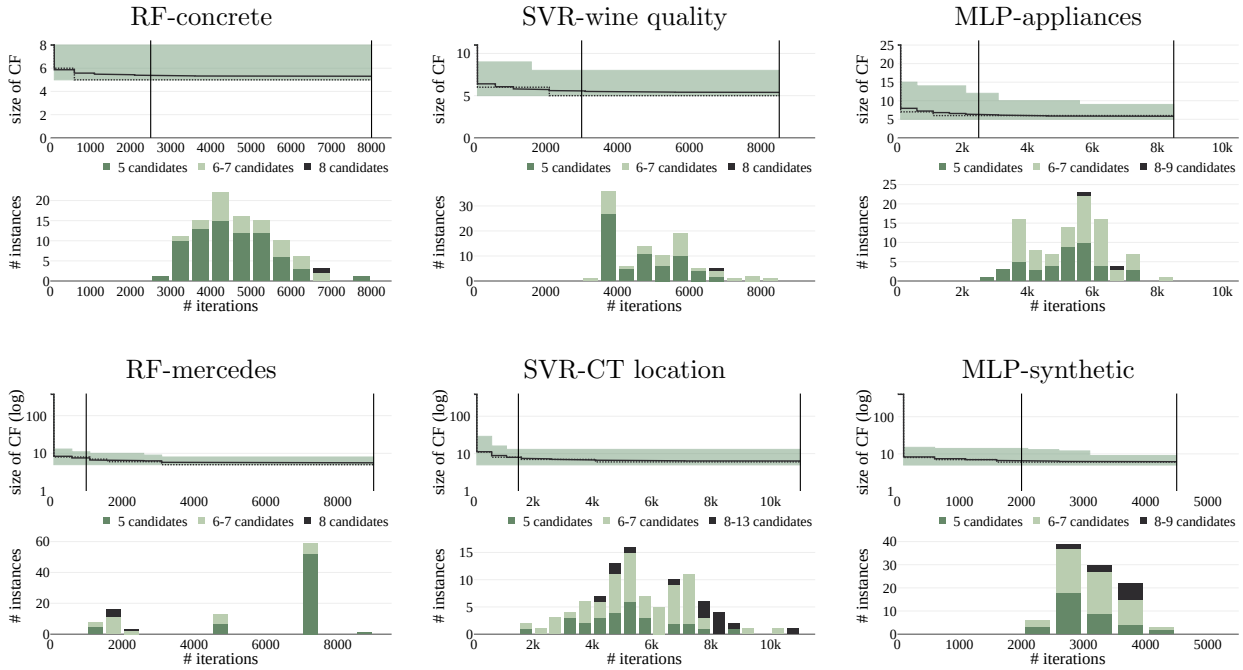

Figure 2: Behavior of TopShap for different models and datasets over 100 instances. TopShap stopped between the two vertical lines. Size of CF are log-scaled for *Mercedes*, *CT location*, and *Synthetic*. Parameters: $k = 5$, confidence $\gamma = 0.95$, warm-up iterations $= 100$.

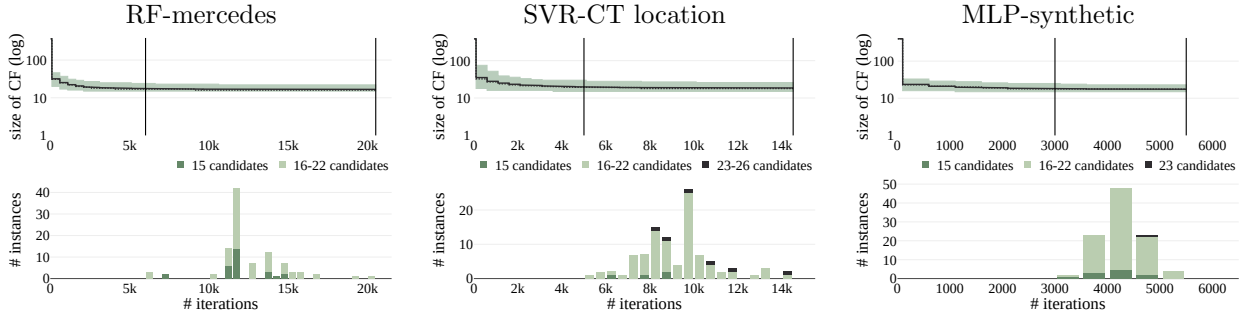

Figure 3: Behavior of TopShap for different models and datasets over 100 instances. TopShap stopped between the two vertical lines. Size of CF are log-scaled. Parameters: $k = 15$, confidence $= 0.95$, warm-up iterations $= 100$.

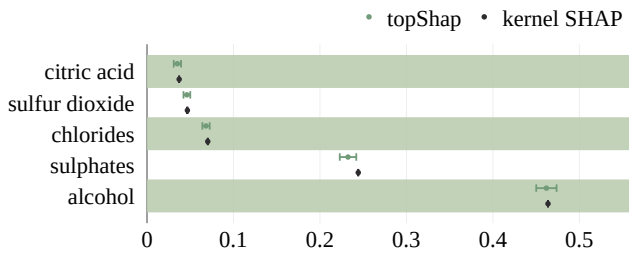

Figure 4: SHAP value approximation of the top-5 features computed by TopShap and Kernel SHAP on the *Wine Quality* dataset for the instance used in Fig. 1a. Values given by TopShap are shown with their confidence intervals.

search with pruning. To be precise, without pruning the size of CF would remain equal to the initial number of features $|F|$ for all iterations. As we have shown here, TopShap started pruning almost immediately after the warm-up, with $|CF|$ quickly decreasing to be close to k. This means that, roughly speaking, TopShap avoided a cost of $(|F|-k) \times modelEvalCost$ per sampling iteration after warm-up, which is an effective cost reduction of the overall process, especially if $k \ll |F|$.

5.5 Comparison with Kernel SHAP

The previous section has shown the gain of incorporating the TopShap pruning within the general approximation process as done by the permutation sampling family of approaches. As presented in Section 3, the other predominant research direction to approximate SHAP values is solving an equivalent WLS problem. In the case of model-agnostic methods, this led to a reference algorithm named Kernel SHAP [2]. In this section, the execution times and the output of TopShap are compared to top-k searches based on Kernel SHAP.

As detailed in [15], the approximation made by Kernel SHAP is also based on sampling, but it is a sampling over the powerset of the features, not a sampling of the permutation of the features. This sampling is performed only once, to build the equations needed to formulate the WLS problem, which is then solved by Kernel SHAP to approximate the SHAP values of all the features. Such an approach implies that reducing the search space is not straightforward for Kernel SHAP when we are only interested in the top-k features. Indeed, for our comparison the only reasonable option was to select these features in a post-processing step after running Kernel SHAP.

Kernel SHAP was called with its default parameters, the number of sampled coalitions in this default setting being equal to two times the number of features plus 2048, and the top 5 features were retained. The TopShap parameters were set to k=5, confidence γ=0.95 and $warmUp$=100. Both methods performed approximations of $f_S(x)$, that is the output of a model f when only a subset S of the features of x are known. To allow for a fair comparison of the global execution time and of the SHAP values themselves, these approximations were made in the same way for both methods using the training dataset as a reference to set values for the unknown part of x.

All times are reported for single-threaded executions on a desktop computer running Ubuntu Linux (2.1 GHz Intel Xeon, 192 GB RAM). The implementation of TopShap is the one provided in the git repository, and the one of Kernel SHAP is KernelExplainer of the SHAP library [28] (version 0.44.0). Both methods are implemented in Python, using numpy vectorization. The models were learnt using scikit-learn and the same models were given to Kernel SHAP and TopShap as input. In both the Kernel SHAP and TopShap implementations, the call to the prediction functions of the models were not made one instance at a time. Each call was made for a batch of instances to amortize the possible overhead due to data structure preparation in prediction functions of the scikit-learn models.

5.5.1 Comparing execution time

For both Kernel SHAP and TopShap, the execution time was measured for the same set of 100 randomly chosen instances of the test dataset. Average execution times per instance are reported in Table II. The time-ratio column corresponds to the average time per instance for Kernel SHAP divided by the one needed by TopShap. Note that executions that took more than one hour and a half (5400 seconds) were stopped. Note also that for *CT location*, one of the largest datasets, Kernel SHAP approximations could not complete due to memory problems and corresponding comparisons could not be reported (Table II).

For nearly all models and datasets, TopShap was much faster than Kernel SHAP. Overall, the run time reduced almost 10-fold (median $\approx$ 9.24), ranging from 2-fold to more than 85-fold. For the smallest dataset, *Concrete Compressive Strength*, TopShap reduced the execution times only in half of the combinations. This dataset contained only 8 features whilst we searched for the top-5. As expected, TopShap could not benefit from a substantial amount of pruning. However, for all other datasets we observed a clear reduction of the execution time.

5.5.2 Comparing SHAP values

We now compare the top-k features and their approximated SHAP values output by both TopShap and Kernel SHAP. The corresponding scores are given in Table II and, as for the execution time, were obtained as average over a set of 100 randomly chosen instances of the test dataset.

TopShap provides confidence intervals and the selection of the top-k is based on these intervals. Thus, it can report more than k features, because of ties due to overlapping intervals. Kernel SHAP does not provide confidence intervals, but, in principle it could report ties if the same SHAP values were obtained for two features. However, this never happened in the experiments discussed here.

dataset	model	Kernel SHAP time (s)	TopShap time (s)	time ratio	Comparative scores recall-like	matching	TopShap # overlaps
concrete	RF	25.63	17.07	1.5	100.0	100.0	0.62
concrete	MLP	18.78	23.73	0.79	100.0	97.8	0.59
concrete	SVR	13.49	12.00	1.12	100.0	99.2	0.54
concrete	STK	51.62	92.37	0.56	100.0	99.8	0.61
wine quality	RF	152.3	17.79	8.56	100.0	97.5	0.81
wine quality	MLP	98.85	40.03	2.47	100.0	100.0	0.92
wine quality	SVR	117.25	14.17	8.28	100.0	100.0	0.62
wine quality	STK	348.62	147.43	2.36	100.0	98.3	1.27
appliance energy	RF	3112.69	35.82	86.9	99.8	90.4	1.52
appliance energy	MLP	479.94	38.17	12.57	99.8	94.0	0.92
appliance energy	SVR	¿ 5400	552.15	¿ 9.78	-	-	8.22
appliance energy	STK	¿ 5400	419.44	¿ 12.87	-	-	2.12
PBMC	RF	201.92	23.23	8.69	99.6	84.8	0.68
PBMC	MLP	645.27	117.77	5.48	100.0	91.5	1.53
PBMC	SVR	242.62	24.47	9.91	100.0	100.0	0.74
PBMC	STK	968.62	367.46	2.64	99.2	87.4	1.49
mercedes	RF	904.45	39.24	23.05	99.4	99.3	1.09
mercedes	MLP	727.6	45.90	15.85	100.0	93.6	1.48
mercedes	SVR	¿ 5400	167.20	¿ 32.30	-	-	1.56
mercedes	STK	¿ 5400	670.74	¿ 8.05	-	-	1.26
CT location	RF	-	86.18	-	-	-	1.75
CT location	MLP	-	353.37	-	-	-	2.90
CT location	SVR	-	2638.82	-	-	-	1.61
CT location	STK	-	5314.55	-	-	-	1.88
synthetic	RF	1892.09	62.57	30.24	99.6	80.8	1.31
synthetic	MLP	927.03	56.15	16.51	100.0	98.8	1.39
synthetic	SVR	684.85	25.30	27.07	100.0	100.0	1.41
synthetic	STK	3276.0	163.07	20.09	100.0	99.0	1.45

Table II: Comparison between Kernel SHAP and TopShap over execution times and SHAP values. Last column: TopShap interval overlaps.

We compared the output of the two methods on three scores. First, we considered Kernel SHAP as a reference and a recall-like score was computed as the percentage of the top-k features obtained with Kernel SHAP that also appeared in the ones reported by TopShap. Table II shows a nearly perfect recall-like score, with only 6 combinations out of 20 for which the average score is not 100% but slightly below. Note that a similar computation of a precision-like score would not be meaningful when Kernel SHAP is taken as a reference. This is because the top-k features of TopShap can include ties due to overlapping confidence intervals, whereas such ties are not taken into account by Kernel SHAP and would not be present in its output.

Next, to assess to which extent the output of the two algorithms agreed for the SHAP values themselves, a matching score was computed. This score was the percentage of the confidence intervals of the top-k features of TopShap that included the SHAP value given by Kernel SHAP for the corresponding feature. An example of a matching score of 4/5 (i.e., 80%) is illustrated in Fig. 4 for an instance taken from the *Wine Quality* dataset. All TopShap intervals encompassed the approximation made by Kernel SHAP, except for the feature *sulphates*, whose value is nevertheless close to the interval's upper bound. The matching scores reported in Table II reflected a strong agreement of both algorithms on the SHAP values obtained.

5.5.3 Confidence interval overlaps

Of course, high matching scores are of interest, if the intervals tend to isolate the SHAP values, i.e., if these intervals have limited or no overlap. This was assessed by counting for each interval the number of non-empty intersections with other intervals. The mean overlap counts are reported in Table II. Most combinations of datasets and models showed a good separation of the SHAP values by their intervals. A few combinations led to a greater number of overlaps, corresponding to cases where several ties were obtained in the output of TopShap. An extreme average of 8.22 is reported in Table II for the *Appliances Energy* dataset and *SVR* model. This was due to a large number of ties, leading to an output containing on average between 10 and 15 features, even if only the top 5 features were requested (Supplementary Materials Fig. S1). Even though ties may seem undesirable at first sight, we emphasize that ties simply mean that several features have a similar importance in the prediction made by the model.

In summary, TopShap and Kernel SHAP agreed well on their approximations of SHAP values. Moreover, the confidence intervals computed by TopShap provided a quantification of the remaining uncertainty. These intervals tended to be rather narrow, implying that, in general, few ties (overlap of intervals between features) were found.

5.6 Use case: the PBMC dataset

A big challenge in biology is to understand how genes interact. We know that understanding gene regulation is key: the question to answer is which transcription factors (TFs, genes that can influence the expression of other genes, including themselves) are responsible for the regulation of a given gene on the genome? Even though we know for a few well-studied cases how TFs regulate a particular gene, generally speaking, within a cell it is unknown which TF is regulating which gene. This, however, can be predicted from data, such as single-cell gene expression data. From such data, regulatory links can be extracted using machine learning and feature importance techniques, in order to reveal which TFs are likely associated to a given target gene. Indeed, if we take for example one of the best currently available workflows, PySCENIC [29], its first computing step consists of learning a tree-based predictor of the expression of the target gene, from which one extracts an ordered list of features (i.e., TFs) through impurity-based feature importances [26]. In this manner, PySCENIC identifies the putative interactions between TFs and genes, from which one can build a single regulatory network for a population of cells.

In a similar way, we applied TopShap to the *PBMC* gene expression dataset, which is commonly used in the single-cell biology domain to demonstrate new methods. After quality control, it consists of 2638 high-quality cells (instances), mainly comprising cell types of the human immune system. One of these cell types is the B cell type. In this use case, we asked a non-trivial biological question regarding B cells: "Which TFs are potentially regulating gene MS4A1, a well-known marker gene for B cells?".

We trained four types of models applying the same experimental setup as for the above reported experiments (see Section 5.3): Random Forest (RF), Multi-Layer Perceptron (MLP), Support Vectors Regressor (SVR), and stacked regressors (STK). Except for STK, all approaches have been previously used on single-cell RNA-seq data [29–31]. After splitting the dataset in training and test sets (respectively 70% and 30% of the 2638 cells) and guaranteeing all cell types were present, we trained the four models.

We then applied TopShap to each of the 105 B cells in the test data and counted how often a TF was mentioned in the top-5 SHAP values (including ties) for the B-cell instances (see Table III). We indicated the five most-occurring top-5 TFs in black font color and we completed the counts for any TF not present in the top-5 of all four models (green font color). We found that 6 out of 8 TFs were either known for their capability to bind in the regulatory genomic regions around MS4A1 or for their involvement in B-cell functioning, as detailed by an entry in the column *Evidence* of Table III.

The transcription factors SPIB and IRF8 were the most frequent features in the top-5 for all models. For the other 6 genes, the four models did not fully agree, suggesting that each regression method provides a different insight into the data. The transcription factors POU2AF1, JUN, and GTF3A were all detected by three methods and only JUN did not show evidence of links with MS4A1 or B cells in the literature. It may play a role, however, as JUN is a pioneer transcription factor involved in general regu-

latory processes of transcription. Alternatively, it might be present due to its involvement in the cellular stress response caused by the experimental procedure of single-cell RNA sequencing. The genes CEBPB, NFATC1, and PAX5 were picked up by only one method each and all could be associated with literature evidence of their role in the regulation of MS4A1 and B cells. One may consider that the variation between the methods is an opportunity to generate new hypotheses for biologists.

For all models, from the viewpoint of algorithm performance, we observed that the width of confidence intervals shrank quickly and that the pruning was (again) very effective (see Fig. 5).

6 Conclusion

We proposed TopShap, a model-agnostic algorithm for searching the k most important features to a prediction. TopShap operates within the SHAP framework to determine local feature importance, the so-called SHAP values. It drastically reduces computational costs by iteratively interleaving sampling steps to improve bounds of SHAP values, and pruning steps to stop any computation for features that can no longer be in the top-k. We demonstrated effectiveness of TopShap by applying it to various datasets, including a use case in the domain of single-cell gene expression analysis. We verified the correctness of its output by comparing our method to the state-of-the-art technique of Kernel SHAP. Moreover, we showed that TopShap can be an order of magnitude faster than Kernel SHAP, if the total number of features in a dataset was much larger than the top-k that one would like to use to explain a model prediction.

The use of TopShap comes with two main limitations. Firstly, it is based on approximations of the SHAP values, as the other state-of-the-art model-agnostic approaches. Secondly, its pruning will not be effective if most features have the same (or very close) SHAP values.

Future work will be directed towards decreasing computational cost further through alternative sampling strategies, e.g., antithetic variates, and investigating complementary pruning based on selection criteria additional to top-k.

References

[1] N. Burkart and M. F. Huber, "A survey on the explainability of supervised machine learning," *Journal of Artificial Intelligence Research*, vol. 70, pp. 245–317, 2021.

[2] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[3] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable AI for trees," *Nature machine intelligence*, vol. 2, no. 1, pp. 56–67, 2020.

[4] D. Wang, S. Thunéll, U. Lindberg, L. Jiang, J. Trygg, and M. Tysklind, "Towards better process management in wastewater treatment plants: Process analytics based on SHAP values for tree-based machine learning methods," *Journal of Environmental Management*, vol. 301, p. 113941, 2022.

[5] L. S. Shapley, "A value for n-person games," in *Contributions to the Theory of Games (AM-28), Volume II*. Princeton University Press, 1953, pp. 307–318.

[6] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[7] E. Strumbelj and I. Kononenko, "An efficient explanation of individual classifications using game theory," *JMLR*, vol. 11, pp. 1–18, 2010.

[8] L. H. B. Olsen, I. K. Glad, M. Jullum, and K. Aas, "A comparative study of methods for estimating conditional Shapley values and when to use them," *arXiv:2305.09536*, 2023.

[9] R. Mitchell, J. Cooper, E. Frank, and G. Holmes, "Sampling permutations for Shapley value estimation," *JMLR*, vol. 23, no. 1, pp. 2082–2127, 2022.

[10] J. Castro, D. Gómez, and J. Tejada, "A polynomial rule for the problem of sharing delay costs in pert networks," *Computers & Operations Research*, vol. 35, no. 7, pp. 2376–2387, 2008.

[11] U. Faigle and W. Kern, "The shapley value for cooperative games under precedence constraints," *International Journal of Game Theory*, vol. 21, pp. 249–266, 1992.

[12] A. Charnes, B. Golany, M. Keane, and J. Rousseau, "Extremal principle solutions of games in characteristic function form: core, chebychev and shapley value generalizations," *Econometrics of planning and efficiency*, pp. 123–133, 1988.

[13] J. Castro, D. Gómez, and J. Tejada, "Polynomial calculation of the Shapley value based on sampling," *Computers & Operations Research*, vol. 36, no. 5, pp. 1726–1730, 2009.

[14] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *KAIS*, vol. 41, no. 3, pp. 647–665, Dec. 2014.

[15] K. Aas, M. Jullum, and A. Løland, "Explaining individual predictions when features are dependent: More accurate approximations to shapley values," *Artificial Intelligence*, vol. 298, p. 103502, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370221000539

Transcription factor	Counts				Evidence
	RF	MLP	SVR	STK	
SPIB	**104**	**71**	**100**	**73**	[32] [33]
IRF8	**93**	**79**	**90**	**82**	[34]
POU2AF1	**57**	21	**63**	**33**	[35] [36]
JUN	5	**49**	**21**	**48**	
GTF3A	**36**	**37**	7	**36**	[37]
CEBPB	16	**27**	8	24	
NFATC1	**41**	6	6	7	[38]
PAX5	12	7	**20**	12	[38]

Table III: Transcription factors (TFs) predicted to regulate MS4A1, a marker gene for B cells. TopShap is run on 105 B cells and *Counts* indicates per model (RF, MLP, SVR, STK) the number of times a TF is in the top-5 of a cell. We selected for each model the top-5 TF with the largest counts (in black, bold font). We completed the table with counts of TFs from other models (in green font). *Evidence* refers to associated literature.

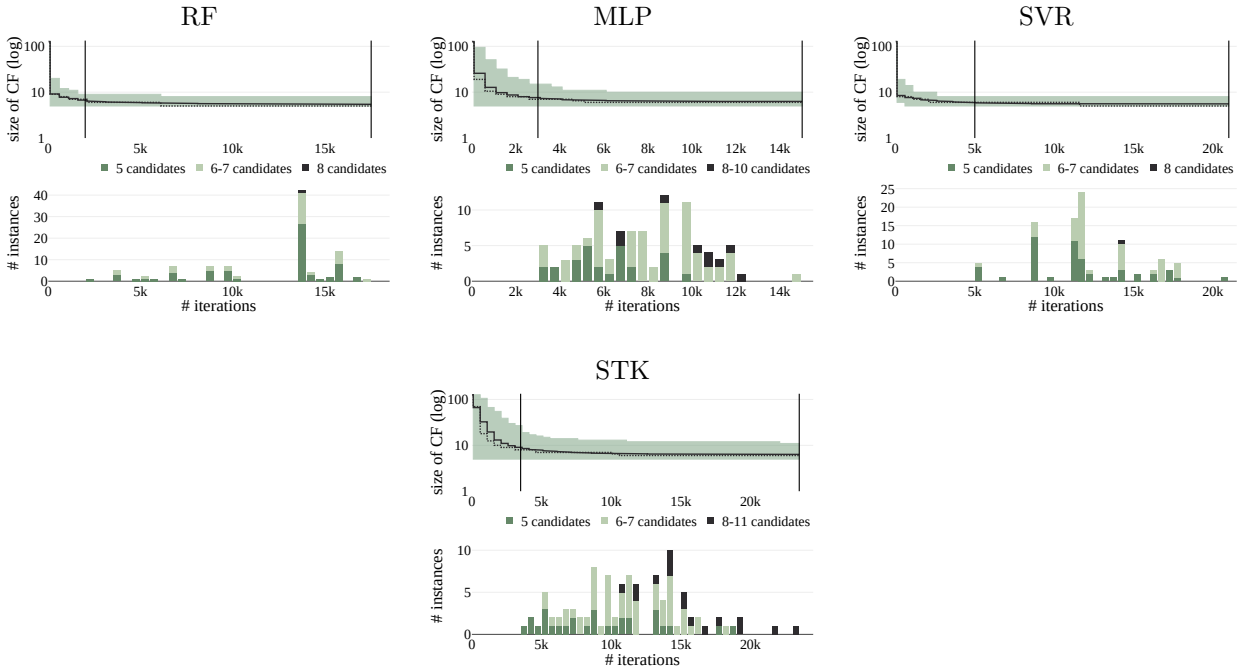

Figure 5: Behavior of TopShap for different models and datasets over 100 instances. TopShap stopped between the two vertical lines. Size of CF are log-scaled. Parameters: $k = 15$, confidence $= 0.95$, warm-up iterations $= 100$.

[16] P. Kolpaczki, V. Bengs, and E. Hüllermeier, "Identifying Top-k Players in Cooperative Games via Shapley Bandits." in *LWDA*, 2021, pp. 133–144.

[17] N. R. Suri and Y. Narahari, "Determining the top-k nodes in social networks using the Shapley value," in *Proc. of the 7th Int. Conf. on Autonomous agents and multiagent systems-Volume 3*, 2008, pp. 1509–1512.

[18] O. J. Dunn, "Multiple comparisons among means," *Journal of the American statistical association*, vol. 56, no. 293, pp. 52–64, 1961.

[19] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[20] I-Cheng Yeh, "Concrete Compressive Strength," 1998. [Online]. Available: https://archive.ics.uci.edu/dataset/165

[21] A. C. Paulo Cortez, "Wine Quality," 2009. [Online]. Available: https://archive.ics.uci.edu/dataset/186

[22] L. Candanedo, "Appliances energy prediction," 2017. [Online]. Available: https://archive.ics.uci.edu/dataset/374

[23] H.-P. K. F. Graf, "Relative location of CT slices on axial axis," 2011. [Online]. Available: https://archive.ics.uci.edu/dataset/206

[24] "UCI Machine Learning Repository." [Online]. Available: https://archive.ics.uci.edu/

[25] "Kaggle: Your Machine Learning and Data Science Community." [Online]. Available: https://www.kaggle.com/

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and others, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[27] "Datasets -Single Cell Gene Expression - Official 10x Genomics Support." [Online]. Available: https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc3k

[28] "SHapley Additive exPlanations." [Online]. Available: https://github.com/shap/shap

[29] B. Van de Sande, C. Flerin, K. Davie, M. De Waegeneer, G. Hulselmans, S. Aibar, R. Seurinck, W. Saelens, R. Cannoodt, Q. Rouchon, and others, "A scalable SCENIC workflow for single-cell gene regulatory network analysis," *Nature Protocols*, vol. 15, no. 7, pp. 2247–2276, 2020.

[30] R. Magnusson, J. N. Tegnér, and M. Gustafsson, "Deep neural network prediction of genome-wide transcriptome signatures – beyond the Black-box," *npj Systems Biology and Applications*, vol. 8, no. 1, p. 9, 2022.

[31] B. Yang, W. Bao, B. Chen, and D. Song, "Single_cell_grn: gene regulatory network identification based on supervised learning method and single-cell rna-seq data," *BioData Mining*, vol. 15, no. 1, pp. 1–18, 2022.

[32] G. H. Su, H. S. Ip, B. S. Cobb, M.-M. Lu, H.-M. Chen, and M. C. Simon, "The Ets protein Spi-B is expressed exclusively in B cells and T cells during development." *The Journal of experimental medicine*, vol. 184, no. 1, pp. 203–214, 1996.

[33] D. Ray, R. Bosselut, J. Ghysdael, M.-G. Mattei, A. Tavitian, and F. Moreau-Gachelin, "Characterization of Spi-B, a transcription factor related to the putative oncoprotein Spi-1/PU.1," *Molecular and cellular biology*, 1992.

[34] L. Grzelak, F. Roesch, A. Vaysse, A. Biton, R. Legendre, F. Porrot, P.-H. Commère, C. Planchais, H. Mouquet, M. Vignuzzi, and others, "IRF8 regulates efficacy of therapeutic anti-CD20 monoclonal antibodies," *European Journal of Immunology*, vol. 52, no. 10, pp. 1648–1661, 2022.

[35] G. Pavlasova and M. Mraz, "The regulation and function of CD20: an "enigma" of B-cell biology and targeted therapy," *Haematologica*, vol. 105, no. 6, p. 1494, 2020.

[36] M. Gstaiger, L. Knoepfel, O. Georgiev, W. Schaffner, and C. M. Hovens, "A B-cell coactivator of octamer-binding transcription factors," *Nature*, vol. 373, no. 6512, pp. 360–362, 1995.

[37] D. Huang and I. Ovcharenko, "Epigenetic and genetic alterations and their influence on gene regulation in chronic lymphocytic leukemia," *BMC genomics*, vol. 18, no. 1, pp. 1–15, 2017.

[38] J. C. Romero-Masters, S. M. Huebner, M. Ohashi, J. A. Bristol, B. E. Benner, E. A. Barlow, G. L. Turk, S. E. Nelson, D. C. Baiu, N. Van Sciver, E. A. Ranheim, J. Gumperz, N. M. Sherer, P. J. Farrell, E. C. Johannsen, and S. C. Kenney, "B cells infected with Type 2 Epstein-Barr virus (EBV) have increased NFATc1/NFATc2 activity and enhanced lytic gene expression in comparison to Type 1 EBV infection," *PLOS Pathogens*, vol. 16, no. 2, p. e1008365, Feb. 2020. [Online]. Available: https://dx.plos.org/10.1371/journal.ppat.1008365

Effective pruning for top-k feature search on the basis of SHAP values Supplementary materials

Lisa Chabrier[1, 2], Anton Crombach[1, 2], Sergio Peignier[3], and Christophe Rigotti[1, 2]

[1]Inria, Lyon Centre, France.
[2]INSA Lyon, CNRS, UCBL, LIRIS, UMR5205, F-69621 Villeurbanne, France.
[3]BF2i, UMR 0203 INRAE, INSA Lyon, France.

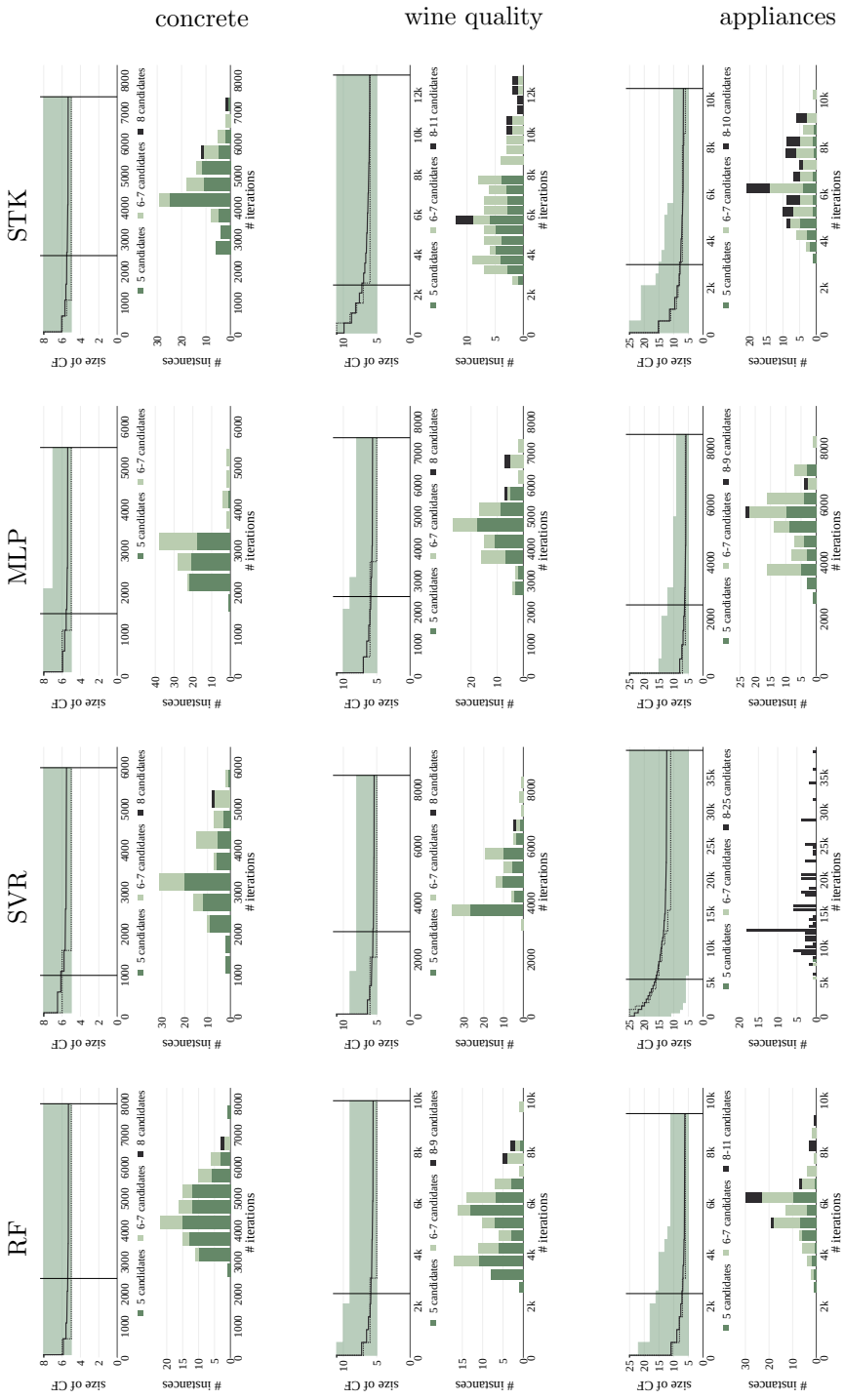

Figure S1: Behavior of TopShap for datasets *Concrete Compressive Strength*, *Wine Quality* and *Appliances Energy* over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 5$, $confidence = 0.95$, warm-up iterations $= 100$.

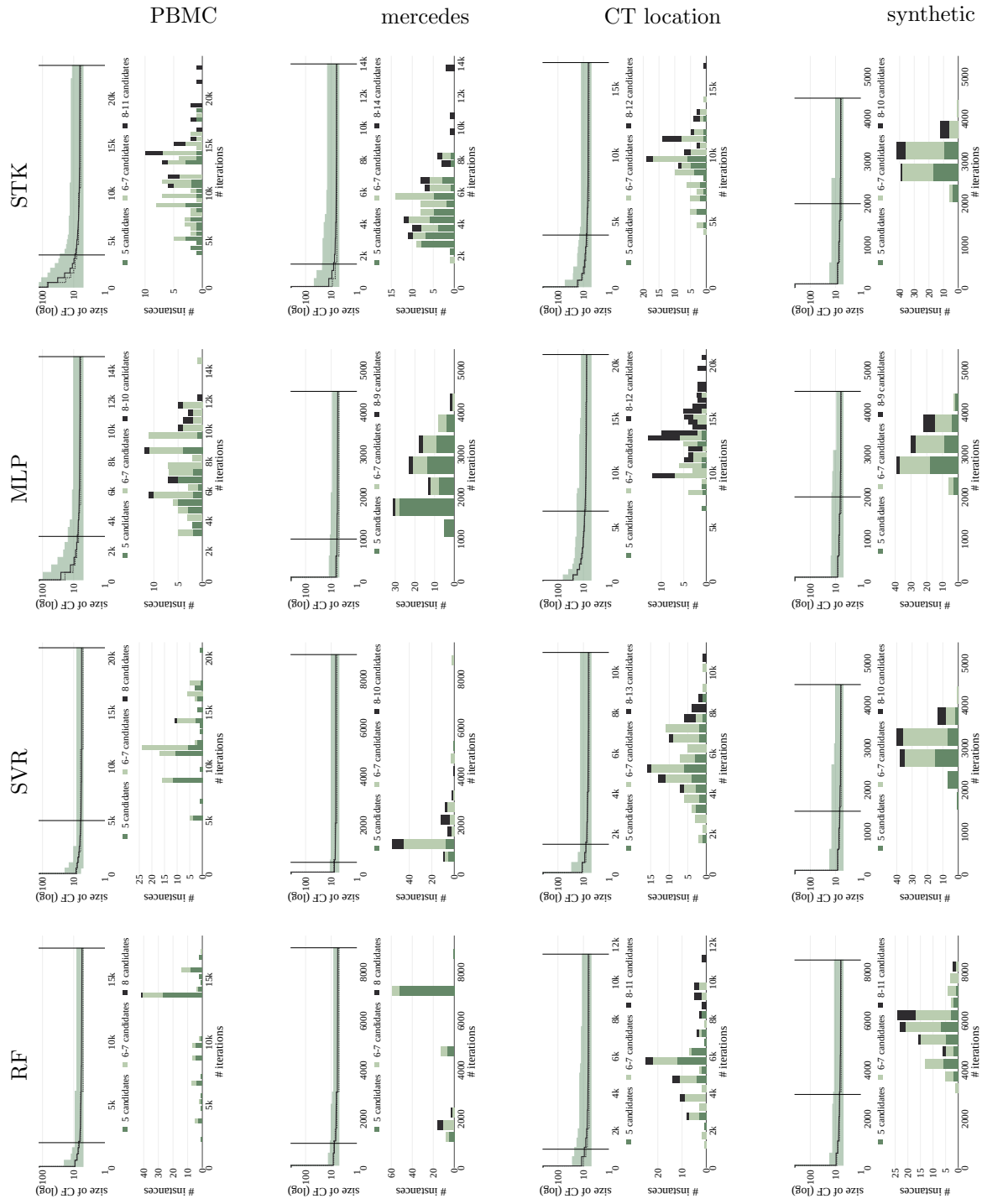

Figure S2: Behavior of TopShap for datasets *PBMC*, *Mercedes*, *CT location*, and *Synthetic* over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 5$, $confidence = 0.95$, warm-up iterations = 100.

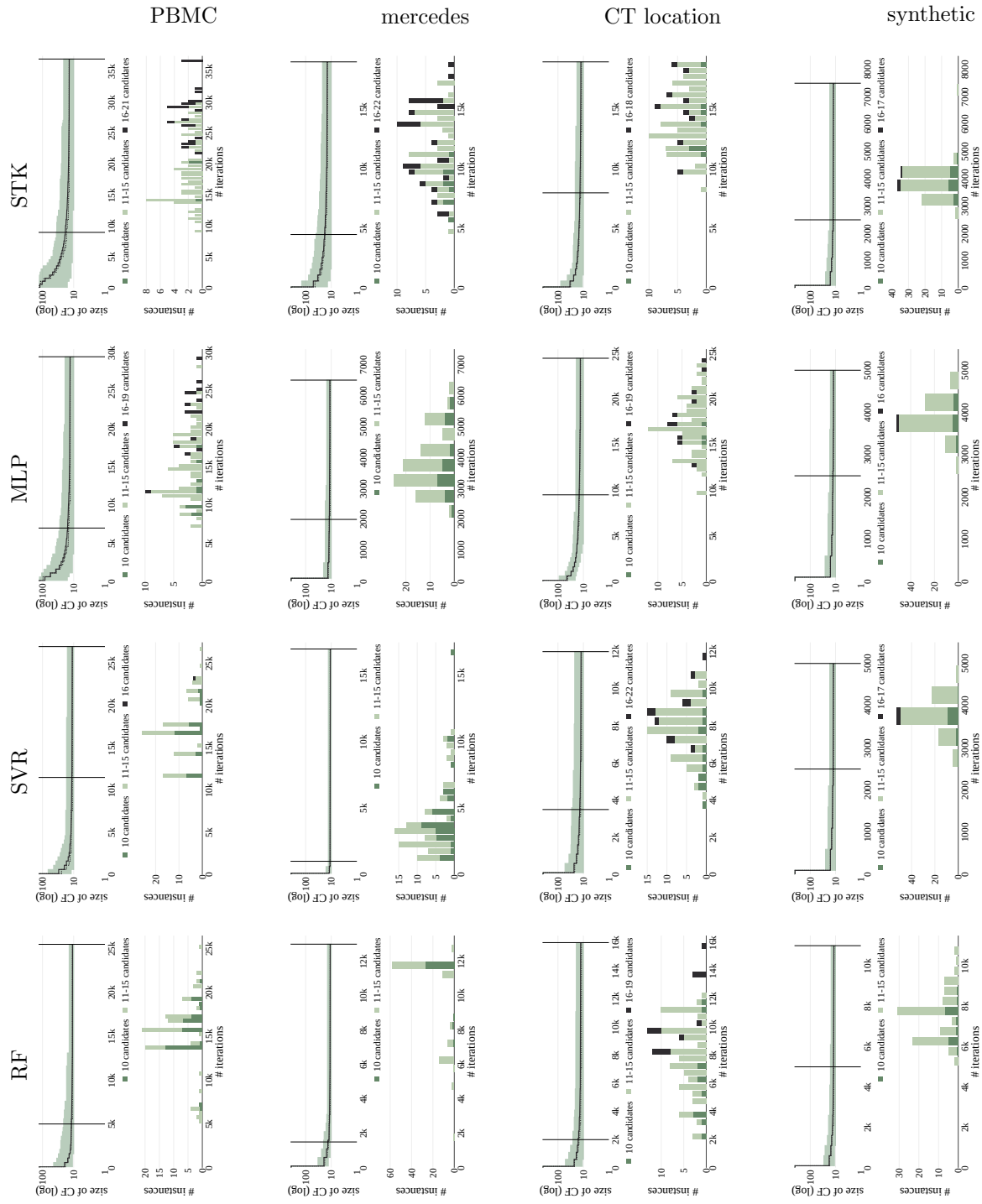

Figure S3: Behavior of TopShap for datasets having more than 25 features (*PBMC*, *Mercedes*, *CT location*, and *Synthetic*) over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 10$, $confidence = 0.95$, warm-up iterations $= 100$.

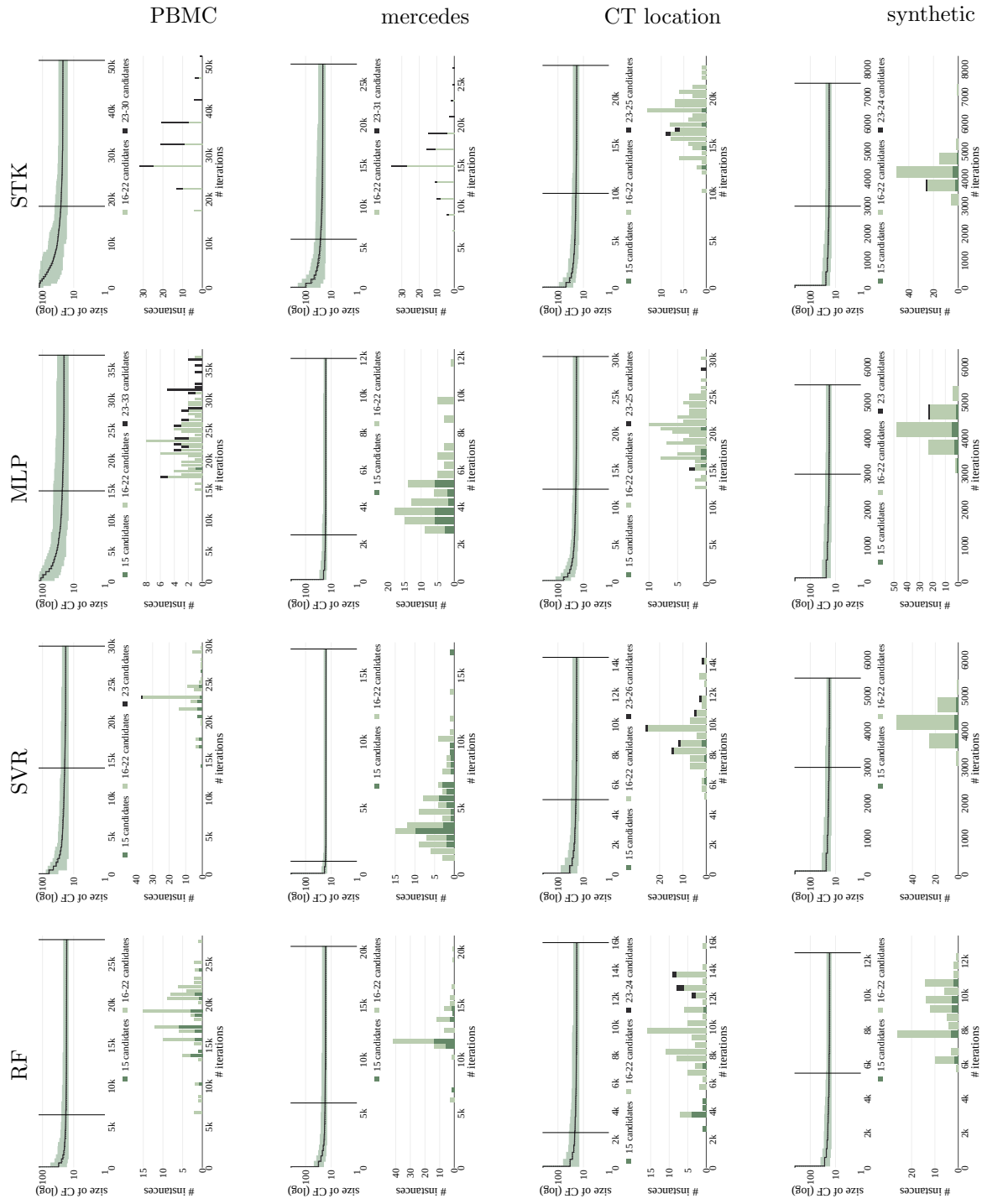

Figure S4: Behavior of TopShap for datasets having more than 25 features (*PBMC*, *Mercedes*, *CT location*, and *Synthetic*) over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 15$, $confidence = 0.95$, warm-up iterations $= 100$.

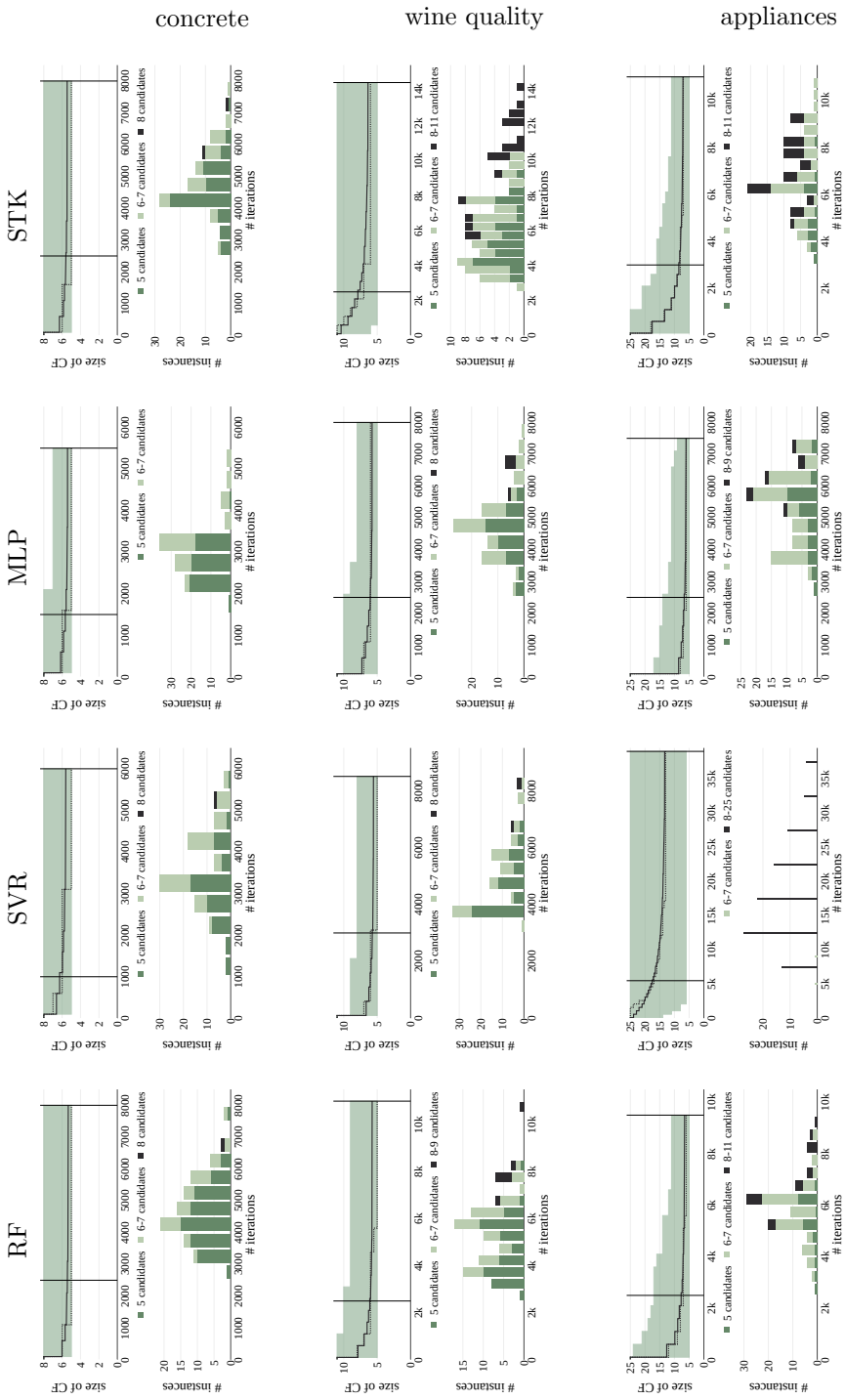

Figure S5: Behavior of TopShap for datasets *Concrete Compressive Strength*, *Wine Quality* and *Appliances Energy* over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 5$, $confidence = 0.99$, warm-up iterations $= 100$.

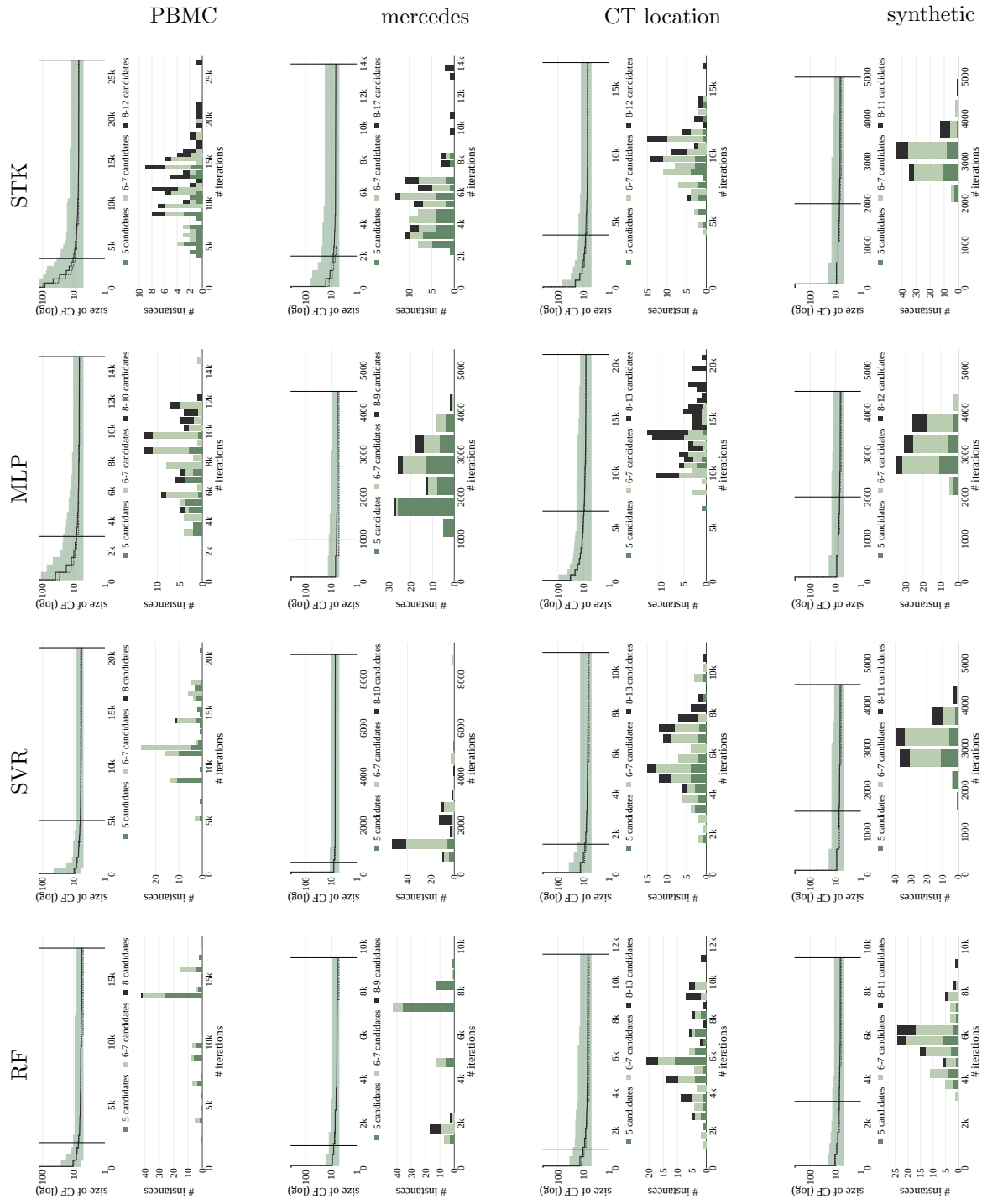

Figure S6: Behavior of TopShap for datasets *PBMC*, *Mercedes*, *CT location*, and *Synthetic* over 100 instances and all models. TopShap stopped between the two vertical lines. Parameters: $k = 5$, $confidence = 0.99$, warm-up iterations = 100.