Rapid Selection of Machine Learning Models Using Greedy Cross Validation

Daniel S. Soper
Department of Information Systems & Decision Sciences,
California State University, Fullerton
dsoper@fullerton.edu

Abstract

This paper introduces a greedy method of performing k-fold cross validation and shows how the proposed greedy method can be used to rapidly identify optimal or near-optimal machine learning (ML) models. Although many methods have been proposed that apply metaheuristic and other search methods to the hyperparameter space as a means of accelerating ML model selection, the cross-validation process itself has been overlooked as a means of rapidly identifying optimal ML models. The current study remedies this oversight by describing a simple, greedy cross validation algorithm and demonstrating that even in its simplest form, the greedy cross validation method can vastly reduce the average time required to identify an optimal or near-optimal ML model within a large set of candidate models. This substantially reduced search time is shown to hold across a variety of different ML algorithms and real-world datasets.

1. Introduction

Organizational development and adoption of artificial intelligence (AI) and machine learning (ML) technologies has exploded in popularity in recent years. One of the most significant drivers of this rapid rise of AI and ML has been cloud computing, through which the vast computational resources required to train and evaluate complex machine learning models have become widely available on an elastic, as-needed basis [1]. Despite the widespread availability of cloud-based computational resources, both the execution time required to train today's complex, state-of-the-art ML models and the cloud computing costs associated with training those models remain major obstacles in many real-world scientific, governmental, and commercial use cases [2]. Further, this problem is often made exponentially worse by the need to perform hyperparameter optimization, wherein a large number of candidate models with varying hyperparameter settings are trained and evaluated in an effort to find the best-performing model [3, 4]. Tools and methods aimed at reducing the computational workload and associated monetary costs of arriving at a final, best-performing ML model are therefore highly desirable.

Since evaluating many ML models can be very time-consuming and expensive, many researchers have considered the important problem of how to find an optimal or near-optimal ML model among the set of all possible models as quickly as possible. Unfortunately, many of the hyperparameters involved in ML model design and training are real-valued, which implies that there is often an infinite number of possible models that theoretically could be evaluated for a particular ML scenario. Recognizing that this situation clearly makes a brute-force model search infeasible, a considerable variety of approaches have been proposed for searching a finite subset of the infinite model space. The simplest and most common of these methods involve performing a grid search or a random search, with the latter approach serving as a natural baseline for inter-method performance comparisons [3]. Several more sophisticated guided search methods have also been proposed, including Bayesian methods [5], population-based approaches such as evolutionary optimization [6], early stopping methods [7], and hypergradient optimization [8, 9], among others. Despite the different rules and theories upon which these guided search methods are based, all share a common general strategy: to identify relationships between hyperparameter values and a performance metric, and then use that knowledge to evaluate models located within promising regions of the search space. This general strategy for performing hyperparameter optimization is illustrated in Figure 1.

With respect to the general approach to hyperparameter optimization, the factors that distinguish one guided search method from another are (1) variations in how tuples of hyperparameters are selected, (2) the stopping conditions that are used, and (3) the information about the relationships between the hyperparameters and the performance metric that is used to guide the search process. Together, these three factors are respectively represented by items *A*, *C*, and *D* in Figure 1. What remains, then, is item *B*, which represents the process of training and evaluating one or more candidate models, with each candidate model corresponding to a tuple of hyperparameter values from

URI: https://hdl.handle.net/10125/80245
978-0-9981331-5-7

item *A*. Evaluating the performance of each candidate model can be accomplished via any statistically defensible process, regardless of the specific guided search method being used. In practice, the task of evaluating an ML model's performance is most commonly carried out using *k*-fold cross validation [10], wherein the data are randomly subdivided into *k* folds, with each fold being iteratively used as a validation set for a model that has been trained using the remaining *k* - 1 folds [11].

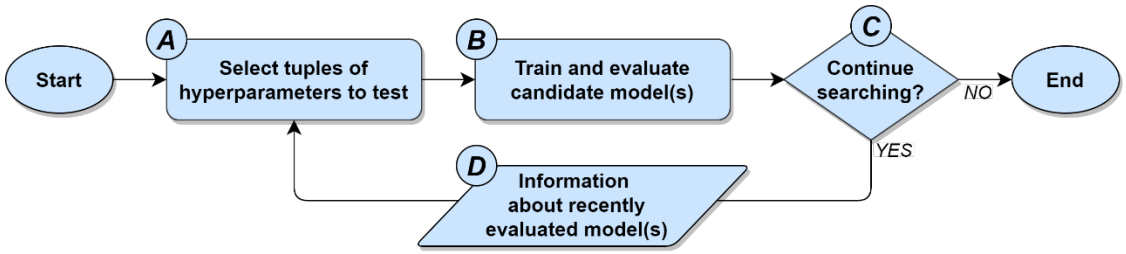

Figure 1. The general approach to hyperparameter optimization used by guided search methods.

In contrast to all existing guided search methods for hyperparameter optimization, this study takes a completely different approach by considering the ML model training and evaluation process as a means of accelerating the search for an optimal model. Put differently, rather than trying to find promising regions within the hyperparameter space, this study instead focuses on the process of measuring model performance as a way of reducing the time and costs associated with evaluating many candidate ML models. In the context of Figure 1, the current study is thus primarily concerned with item *B*, which, as noted in the discussion above, has been generally overlooked as a means of performing rapid hyperparameter optimization. Given that ML model performance is most commonly carried out using *k*-fold cross validation, this paper explicitly seeks to pioneer a new approach to hyperparameter optimization by inquiring into the following general research question:

Research Question: When performing hyperparameter optimization with *k*-fold cross vali-dation, is it possible to improve the average time required to find the best-performing model by taking a greedy approach to the *k*-fold process itself?

The balance of this paper is organized as follows: Section 2 provides a review of the related literature by describing current methods of performing hyperparameter optimization, as well as the standard approach to *k*-fold cross validation. The greedy *k*-fold cross validation algorithm that forms the core of the current study is introduced in Section 3, along with a discussion of the algorithm's properties. Section 4 describes a series of experiments that were undertaken to evaluate the performance of the greedy *k*-fold method relative to the baseline standard *k*-fold method, with the experiments comparing the ML model search performance of the greedy and standard methods across a variety of different ML algorithms and real-world datasets. The outcomes of the evaluative experiments are presented and discussed in Section 5, with the results indicating that greedy *k*-fold cross validation can vastly reduce the average time required to identify the best-performing ML model within a set of candidate models. The paper concludes with Section 6, which provides a brief summary, describes the limitations of the work, and offers a few remarks about future research.

2. Related work

2.1 Hyperparameter optimization & model selection

When developing ML-based solutions, it is standard practice to evaluate a variety of different ML models with a view toward identifying the best model possible given the project's temporal or financial constraints (which collectively define the ML project's computation budget) [12]. The term *model* as it is being used here refers to a combination of an ML algorithm and the specific values that have been chosen for the algorithm's tunable or definable parameters. These parameters can be structural – for example, the number of hidden layers or the number of nodes per layer in a neural network – or they can be algorithmic parameters that control the learning process, such as the mini-batch size or the learning rate. Collectively, these structural and algorithmic parameters are referred to as the model's *hyperparameters*, and the task of searching for the best possible combination of hyperparameter settings for a particular problem is referred to as *hyperparameter optimization* [3]. Evaluating many combinations of ML algorithms and hyperparameter settings (i.e., evaluating many models) is typically necessary because research has shown that no single ML algorithm or set of hyperparameter settings yields optimal results for all possible datasets or problem domains [13, 14]. Indeed, a particular combination of an ML algorithm and a set of hyperparameter settings may perform very well in one scenario while performing

very poorly in another scenario. Since every ML model has hyperparameters, and since identifying the best possible model given the project's computational budget is commonly of paramount importance, hyperparameter optimization has become an indispensable step in the broader process of developing an ML-based solution.

2.2 Current hyperparameter optimization methods

2.2.1 Grid search. Grid search is one of the most widely used and well-established methods of performing hyperparameter optimization among machine learning practitioners [15]. In a grid search, the ML practitioner first specifies a finite set of values for each hyperparameter, after which the grid search algorithm performs an exhaustive search by evaluating the Cartesian product of these sets of hyperparameter values [3]. As with all hyperparameter optimization methods, the ML practitioner must instruct the grid search algorithm to use a specific performance metric when evaluating a set of candidate models, with overall model performance typically being determined via *k*-fold cross validation [10].

2.2.2 Random search. Rather than iterating over the Cartesian product of all of the sets of hyperparameter values defined by the ML practitioner, a random search proceeds by evaluating ML models whose hyperparameter values have been chosen randomly. As with a grid search, the random search method can be readily applied to discrete, continuous, or mixed hyperparameter spaces. It is common practice when conducting a random search to establish a computational budget for the search process in which the search for the best-performing model continues until a certain number of models have been evaluated or a certain amount of time has elapsed [3].

There are several reasons why the random search method serves as an excellent baseline against which to compare the performance of other hyperparameter optimization methods. First, the random search method does not make any assumptions about the specific ML problem domain or hyperparameter space in which it is operating. Similarly, the random search method does not make or rely on any assumptions about the specific ML algorithm whose hyperparameters it is attempting to optimize. Lastly, given a sufficient computational budget, a random search will eventually identify a model in the hyperparameter space whose distance from the globally optimal model is within any arbitrarily chosen degree of precision. Since random search has been shown to outperform many sophisticated search algorithms in scenarios involving a fixed computational budget and no prior knowledge of the hyperparameter space, using the random search performance as a comparative baseline is critical when evaluating the new metaheuristic search algorithms [16].

2.2.3 Bayesian optimization. In the context of hyperparameter optimization, Bayesian optimization is an iterative method that relies on Bayes's Theorem to guide the hyperparameter search process [17]. This method works via a combination of two primary elements: (1) a probabilistic surrogate model, and (2) an acquisition function that is based on the surrogate model [3]. During each iteration, the surrogate model is first updated using the actual observations about the relationship between the hyperparameters and the performance metric that have thus far been obtained, yielding a posterior distribution. The acquisition function is then maximized to identify the most promising tuple of hyperparameter values to evaluate next. A candidate model that uses the most promising tuple of hyperparameter values is then evaluated in the actual search space, with the results being used to update the surrogate model for the next iteration. This process repeats until a stopping condition is met, such as the exhaustion of a computational budget or a sufficiently small difference in candidate model performance from one iteration to the next.

2.2.4 Evolutionary optimization. As the name suggests, evolutionary optimization is an iterative method of performing hyperparameter optimization that relies on principles adopted from the biological process of evolution, such as mutation, recombination, adaptation to the environment, and survival of the fittest [18]. The evolutionary optimization process begins by creating a population consisting of a reasonably large number of randomly generated hyperparameter configurations. Next, the performance of each of these members of the population is evaluated in light of the data and the chosen ML algorithm, typically by means of *k*-fold cross validation. The tuples of hyperparameter values are then ranked according to their observed levels of performance. Next, the worst-performing members of the population are discarded and replaced by new members, with the hyperparameter values for the new members being generated by means of mutation or recombination of the hyperparameter values of the best-performing members of the population. Finally, the performance of each of the newly generated members is evaluated, and the population is re-ranked. These tasks are repeated until a stopping condition is met, such as the exhaustion of a computational budget or a sufficiently small difference in the performance of the best-performing candidate model from one iteration to the next.

2.2.5 Early stopping optimization. Early stopping optimization is an approach to hyperparameter optimization that relies on a strategy of pruning many unpromising hyperparameter configurations as quickly as possible, thus allowing a steadily increasing proportion of the available computational budget to be directed at evaluating more promising hyperparameter configurations in greater detail. Several variants of the early stopping method have been proposed in recent years, notably including successive halving [19, 20], asynchronous successive halving [4], and Hyperband [7]. Beginning with a fixed computational budget and a large, randomly generated set of candidate models, the early stopping methods first perform a quick, shallow evaluation of each candidate model. Next, the worst-performing models are discarded (i.e., any further consideration of the worst-performing models is stopped early), and a larger proportion of the computational budget is allocated toward evaluating the remaining candidate models in greater detail. This process is repeated until only a single candidate model remains.

2.2.6 Hypergradient optimization. Hypergradient optimization is a general term for the collection of methods that perform hyperparameter optimization by computing a gradient with respect to a ML model's hyperparameters (i.e., a *hypergradient*) and then updating the hyperparameter values by using gradient descent [8, 9, 21-23]. Gradient descent and its variants have been used for several decades as a basis for computing elementary parameter values for a wide variety of ML algorithms [24-26]. Rather than using gradient descent exclusively for the purpose of optimizing a machine learning model's elementary parameters, however, in hypergradient optimization gradient descent is leveraged as a means of optimizing the model's hyperparameters, as well.

2.2.7 Summary of current hyperparameter optimization methods. Grid search and random search notwithstanding, the conceptual paradigm employed by the other existing methods of model selection and hyperparameter optimization described above is to focus on the relationship between the values of the hyperparameters and the values of the metric that is being used to evaluate the performance of each candidate model. These hyperparameter optimization methods assume the presence of an underlying but unknown objective function that maps the hyperparameter values for the current ML algorithm to the value of the performance metric. The general goal of such methods, then, is to accumulate information about the nature of the objective function, and then exploit that information to select hyperparameter values that will yield a model whose performance is as close to the global optimum as possible. The greedy *k*-fold cross validation algorithm described in Section 3 differs from all of these guided search methods in that it does not actively compute or choose new hyperparameter values to evaluate as the optimization process unfolds. Instead, the greedy *k*-fold method exploits the model evaluation process itself as a means of accelerating the hyperparameter optimization task.

2.3 *k*-fold cross validation & ML model selection

Broadly, *k*-fold cross validation is a technique for judging how well a model will generalize to scenarios involving novel data that were not considered or "seen" when the model was being trained [27, 28]. In the context of machine learning, *k*-fold cross validation has become the primary method used by ML practitioners when evaluating candidate models [10], not only because of the method's utility as an estimator of generalization performance, but also because of its ability to reveal problems with selection bias and overfitting [29]. The standard *k*-fold process involves splitting the data into *k* subsets of approximately equal size (called *folds*), with each fold being iteratively used as a validation set for a candidate model that has been trained using the data from the remaining $k - 1$ folds [11]. During each evaluative iteration, $k - 1$ of the folds are thus used to train the candidate model, with the model's performance being measured using the remaining fold. This process is repeated until each fold has been used exactly once as a validation set, yielding a total of *k* iterations of training and validation for each candidate model. Finally, the model's overall performance is estimated as the mean of the performance values obtained from each evaluative iteration. This standard method of performing *k*-fold cross validation is illustrated in Figure 2.

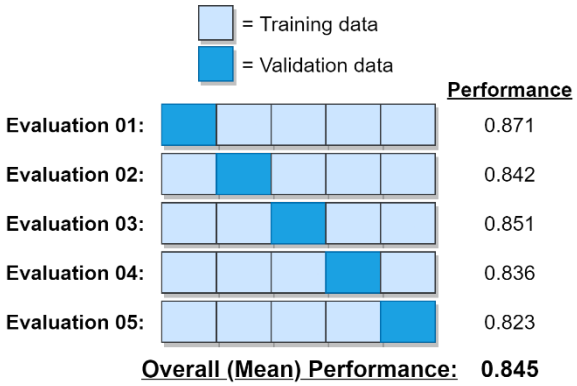

Figure 2. Standard *k*-fold cross validation (with $k = 5$ folds).

2.4 Model selection

Given a specific problem scenario and machine learning algorithm, the most widely used strategy for identifying and selecting the best-performing ML model is to conduct hyperparameter optimization using k-fold cross validation [29, 30]. The set of n candidate models to be evaluated may be defined in advance (e.g., when using a grid search or a random search) or may be defined dynamically as the model search process unfolds (e.g., when using one of the guided search methods described previously). After evaluating as many models as possible given the constraints of the computational budget, the candidate model with the most desirable overall performance characteristics is chosen as the final model. A graphical representation of the cross validation-based ML model selection process is shown in Figure 3.

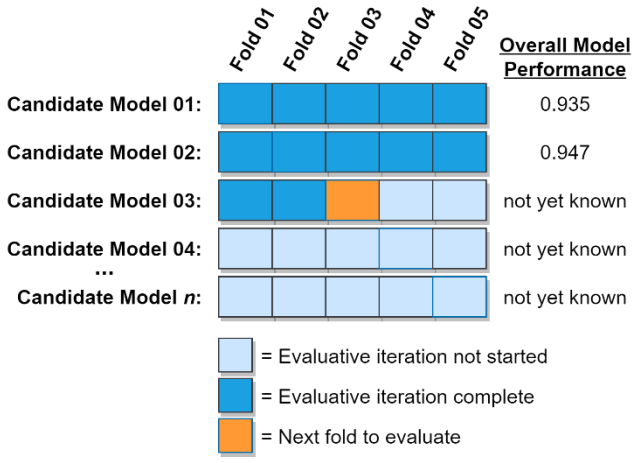

Figure 3. ML model selection using standard *k*-fold cross validation (with *n* candidate models and *k* = 5 folds).

As shown in the figure above, the combination of n candidate models and k evaluative iterations can be thought of as a table, with each column representing one of the k evaluative iterations and each row representing a candidate model. The overall progress that has been made toward completing the evaluation of each candidate model is indicated by the color of the table cells, with dark-colored cells indicating that the evaluative iteration has been completed for the corresponding fold and model. Evaluation of candidate models thus proceeds one fold at a time, from left to right, top to bottom until the computational budget has been exhausted, at which time the candidate model with the best overall performance is chosen as the final model. Using this conceptual framework, it is convenient to discuss the total amount of work involved in the ML model selection process in terms of the number of folds evaluated, where a "fold evaluation" refers to a fold being used to validate a candidate model that has been trained with the remaining folds. The maximum number of folds that could possibly be evaluated in the ML model selection process is thus nk, with reasonable values for a computational budget b falling in the interval $k \leq b \leq nk$.

3. Greedy *k*-fold cross validation

Having briefly described the standard k-fold cross validation and ML model selection processes as well as the existing approaches to hyperparameter optimization, we are now fully equipped to understand the greedy k-fold cross validation algorithm introduced in this section. Whereas all existing guided search methods seek to accelerate the ML hyperparameter optimization and model selection process by identifying and searching promising areas within the hyperparameter space, the greedy k-fold method takes a completely different approach by instead focusing on the k-fold cross validation process itself as a means of achieving rapid hyperparameter optimization and model selection.

At a fundamental level, the greedy k-fold cross validation method proposed here differs from the standard k-fold cross validation process in just one important way. In the standard approach, all of the folds for a given ML model are considered as validation sets in sequential order, one after another, thus allowing the overall performance of the model to be computed before the algorithm moves on to the next candidate model (*vide supra*, Figure 3). By contrast, greedy k-fold cross validation considers a sequence of folds that may originate from *different* ML models, with the specific model and validation fold to evaluate next being greedily chosen at runtime. Put differently, the standard approach to k-fold cross validation can be thought of as relying on a sequence of *within-model* fold evaluations, while the greedy approach can be thought of as relying on a sequence of *between-model* fold evaluations.

In its simplest form, the greedy k-fold cross validation algorithm begins by obtaining an incomplete performance estimate for each candidate model by using just the first fold as a validation set, after having trained the model using the k - 1 remaining folds. The model with the best initial performance is then identified, after which the second fold for that model is used as a validation set (with the remaining folds naturally being used as the training set). The performance estimate for the model is then updated to reflect the mean performance thus far observed after having tested the model using the first two folds as validation sets. The incompletely evaluated model with the best mean performance at that moment is then identified, after which its next available fold is used as a validation set

and the model's mean performance is updated. This process repeats until the computational budget has been exhausted, after which the algorithm returns the best, fully evaluated model. A graphical example of greedy *k*-fold cross validation in progress is shown in Figure 4 below, followed by a complete description of the ML model selection process as performed using greedy *k*-fold cross validation in Algorithm 1.

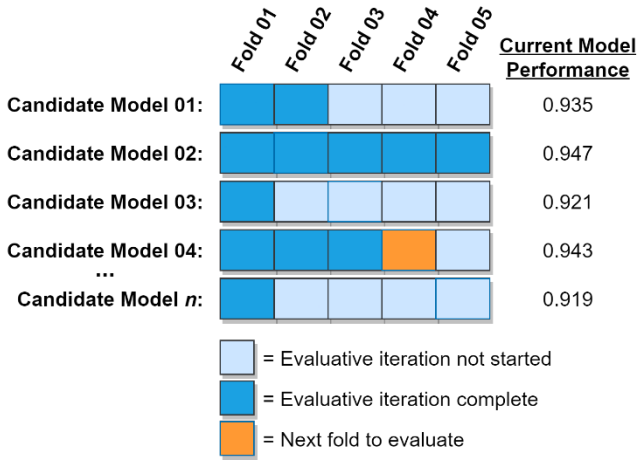

Figure 4. Greedy k-fold cross validation in progress.

Algorithm 1. ML model selection using greedy *k*-fold cross validation.

Input: M (set of candidate models), k (number of folds), D (dataset), b (computational budget)
Output: Best-performing, fully evaluated model

split D into k folds, s.t. $D = \{d_1, d_2, ..., d_k\}$
$\eta \leftarrow 0$ (number of fold evaluations completed)
for each $m \in M$ **do**
  train m using folds $\{d_2, ..., d_k\}$
  $P_m \leftarrow$ performance of m evaluated using fold d_1
  $F_m \leftarrow 1$ (number of folds evaluated for m)
  $\eta \leftarrow \eta + 1$
end for
while $\eta < b$ **do** (while the computational budget is not exhausted)
  $m^* =$ best incompletely evaluated $m \in M$ (given the current mean performance for each m, per P)
  $F_{m^*} \leftarrow F_{m^*} + 1$
  train m^* using all folds $d_i \in D$ where $i \neq F_{m^*}$
  evaluate performance of m^* using fold $d_{F_{m^*}}$
  $P_{m^*} \leftarrow$ mean performance of m^* for folds $\{d_1, ..., d_{F_{m^*}}\}$
  $\eta \leftarrow \eta + 1$
end while
return best, fully evaluated $m \in M$, per P

As indicated in the *while* loop, the greedy *k*-fold cross validation algorithm behaves greedily by always pursuing the most promising available option, with the extent to which an option is promising being determined by the current mean performance of its corresponding model. Put differently, the next fold that the greedy algorithm will evaluate will always originate from the best incompletely evaluated model, as determined by each candidate model's current mean performance. In this way, the greedy *k*-fold cross validation algorithm focuses its early efforts on the most promising candidate models. As time passes and the most promising models become fully evaluated, the algorithm will steadily evaluate folds from less and less promising models, but will never waver from the principle of greedily pursuing the most promising of its available options on each iteration. This behavior thus increases the probability of an optimal or near-optimal model being identified before the computational budget is exhausted.

4. Evaluative experiments

Having presented the greedy *k*-fold cross validation algorithm in the previous section, we will next describe a set of evaluative experiments that were undertaken to assess the algorithm's performance in comparison to that of the standard *k*-fold method. To rigorously evaluate the greedy *k*-fold method, its performance was compared to the standard method under many experimentally manipulated conditions. including using a variety of different ML algorithms, a variety of real-world datasets, and varying values of the number of folds (k) and the number of candidate models (n). Prior to describing these experiments, however, it is necessary to define the way in which the performance of the greedy and standard *k*-fold cross validation methods was measured. First, recall from the earlier discussion that for a hyperparameter space containing n models that are tested with cross validation using k folds, the total number of fold evaluations is nk. For a set of n candidate models, then, the performance of each method was measured by the average search time required to find the best-performing model in the set, with search time being calculated as the ratio of the total number of fold evaluations that were required to find the best-performing model relative to the total number of nk possible fold evaluations. This is a very convenient measure of search time since it naturally yields an interval that ranges from 0.0 to 1.0, thus allowing straightforward performance comparisons to be made across experimental conditions.

Given the search time metric described above, it can be readily calculated that the time required to fully evaluate each candidate model is $\frac{k}{nk} = \frac{1}{n}$. The maximum

theoretical time required to find the optimal model using the standard *k*-fold method is thus $\frac{nk}{nk} = 1.0$, while the minimum search time for the standard method is $\frac{1}{n}$. Since the standard *k*-fold method essentially employs a linear search strategy, the average theoretical search time for the standard method is $\frac{1}{nk} \cdot \frac{nk}{2} = \frac{1}{2}$. On average, then, the standard *k*-fold cross validation method can be expected to find the optimal model after having fully evaluated 50% of the candidate models. While the maximum theoretical search time for the greedy *k*-fold method is also 1.0, the minimum time for the greedy method to find the optimal model is $\frac{n+k-1}{nk}$. The average theoretical search time for the greedy method will depend on the distributional properties of the training data and will hence vary from one scenario to the next.

As noted previously, a variety of ML algorithms and real-world datasets were used in the experiments to compare the performance of the greedy *k*-fold and standard (baseline) *k*-fold cross validation methods. The algorithms included the Bernoulli Naïve Bayes, Decision Tree, and K-Nearest Neighbors (KNN) classifiers, each of which was chosen because of its distinct approach to performing the classification task. All of the ML algorithms used in the experiments are open-source and freely available via the Python *scikit-learn* library [31]. The three datasets used in the experiments are all well-known among ML practitioners, and included the Wisconsin Diagnostic Breast Cancer dataset (569 cases, 30 features, 2 classes), the Boston Home Prices dataset (506 cases, 13 features, 4 classes – discretized using a quartile split), and the Optical Recognition of Handwritten Digits dataset (1797 cases, 64 features, 10 classes). These three datasets were chosen because they varied widely in terms of their numbers of cases, features, and classes, and because they are all freely available as part of the *scikit-learn* library [31], thus helping to ensure that the results can be easily replicated.

For each combination of ML classifier and dataset, n different models were evaluated using $k \in \{5, 10, 20\}$ folds for each *k*-fold method. These values of k were chosen based on their common usage in applied ML projects. The set of values used in the experiments for *n* was derived from a geometric sequence with a common factor of two: $f(x) = 2^x$ for $x \in \{7, 8, \ldots, 11\}$, yielding: $n \in \{128, 256, 512, 1024, 2048\}$.

The candidate models that were evaluated in the experiments varied according to the values of their hyperparameters, with the hyperparameter settings for each model being chosen randomly in accordance with Bergstra & Bengio [15] using the same ranges of possible values for each hyperparameter that were used by Olsen et al. [14]. For each combination of classifier, dataset, and *n*, the same set of candidate models was used to evaluate the standard (baseline) *k*-fold method and the greedy *k*-fold method. This approach was adopted to ensure that any differences in inter-method performance could not be attributed to variation in hyperparameter settings among the *n* available models. Finally, 30 iterations of each experiment were carried out for each combination of classifier, dataset, *n*, and *k* in order to ensure that the resulting performance metric distributions would be statistically stable. With two *k*-fold methods, three ML algorithms, three datasets, three values of *k*, and five values of *n*, a total of 270 different conditions were tested throughout the course of the experiments. The results of these efforts are reported and discussed in the following section.

5. Results & discussion

The average search time required by the greedy and standard *k*-fold cross validation methods to find the optimal model using different ML algorithms, datasets, and values of *k* is provided in Table 1 below, with the results in the table being computed using all possible values of *n*.

Table 1. Average search time to find optimal model.

Dataset	Algorithm	Greedy *k*-Fold Method			Standard *k*-Fold Method		
		$k = 5$	$k = 10$	$k = 20$	$k = 5$	$k = 10$	$k = 20$
Boston Home Prices	Bernoulli Naïve Bayes	0.342^{***}	0.299^{***}	0.301^{***}	0.495	0.496	0.496
	Decision Tree	0.280^{***}	0.231^{***}	0.229^{***}	0.522	0.524	0.514
	K-Nearest Neighbors	0.320^{***}	0.291^{***}	0.278^{***}	0.501	0.488	0.489
Wisconsin Diagnostic Breast Cancer	Bernoulli Naïve Bayes	0.282^{***}	0.217^{***}	0.212^{***}	0.491	0.521	0.488
	Decision Tree	0.291^{***}	0.248^{***}	0.219^{***}	0.516	0.547	0.492
	K-Nearest Neighbors	0.328^{***}	0.283^{***}	0.306^{***}	0.459	0.493	0.492
Optical Recognition of Handwritten Digits	Bernoulli Naïve Bayes	0.236^{***}	0.164^{***}	0.146^{***}	0.523	0.515	0.468
	Decision Tree	0.231^{***}	0.148^{***}	0.113^{***}	0.494	0.440	0.526
	K-Nearest Neighbors	0.270^{***}	0.193^{***}	0.191^{***}	0.504	0.482	0.530

*** indicates $p < 0.001$ for a Welch's t-test comparing the performance of the greedy method to the standard method

The probability values in Table 1 are for two-tailed Welch's *t*-tests comparing the performance of the greedy method to the standard method in corresponding experimental conditions. Welch's *t*-tests were used because there was no reason to expect that the distributions of the performance metrics for the greedy and standard *k*-fold methods would have equal variances, and unlike many other types of *t*-tests, Welch's *t*-tests allow for unequal variances between the independent samples [32]. As the probability values in the table reveal, in terms of its ability to quickly identify the optimal model, the greedy *k*-fold method statistically outperformed the standard *k*-fold method at the $p < 0.001$ level in every combination of dataset, ML algorithm, and value of *k* used in the experiments. These results provide strong statistical evidence for the superiority of the greedy *k*-fold method over the standard *k*-fold method in identifying optimal or near-optimal ML models when operating under the constraint of a computational budget.

Having established the statistical superiority of the greedy *k*-fold method, we may next inquire into the relative magnitude of that superiority. Among the 27 unique combinations of datasets, ML algorithms, and values of *k* reported in Table 1, the overall mean search time for the greedy method was 0.246 (std dev = 0.059), while the overall mean search time for the standard method was 0.500 (std dev = 0.023). This suggests that among the datasets and ML algorithms used in the experiments, the greedy method on average identified the optimal model among the set of *n* candidate models after completing 24.6% of the possible fold evaluations, while the standard method on average identified the optimal model after completing 50.0% of the possible fold evaluations. Note that this latter outcome conforms precisely with the theoretically expected average for the standard method described in Section 4. Put differently, the greedy *k*-fold method identified the best-performing ML model among the set of candidate models *more than twice as quickly* on average than the standard *k*-fold method. As an illustrative example of this major difference in performance, Figure 5 below depicts the average search time of the greedy vs. standard *k*-fold methods using a decision tree classifier on the Wisconsin Diagnostic Breast Cancer dataset.

The results presented in Table 1 and Figure 5 reflect the comparative performance of the standard and greedy *k*-fold methods across a variety of datasets, ML algorithms, and values of *k*. Those results, however, were computed for all of the possible values of *n* that were used in the experiments. It is, of course, possible to gain more detailed insights by disaggregating these results and considering how various values of *n* impact the comparative performance of the standard and greedy *k*-fold methods. While space limitations make it infeasible to visualize the comparative performance of these two different cross validation methods for all 135 unique combinations of datasets, ML algorithms, values of *k*, and values of *n* used in the experiments, a representative example is provided in Figure 6 below. This figure shows how the greedy *k*-fold method performed against the standard (baseline) method for varying numbers of *n* candidate models on the Boston House Prices dataset at values of $k \in \{5, 10, 20\}$.

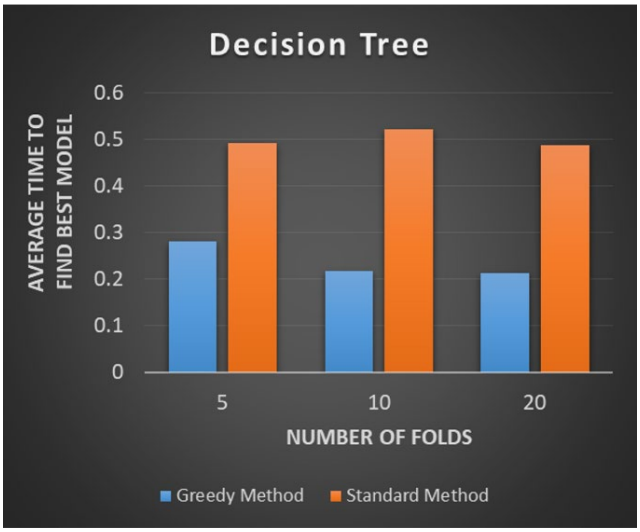

Figure 5. Comparative performance of the greedy and standard *k*-fold methods.

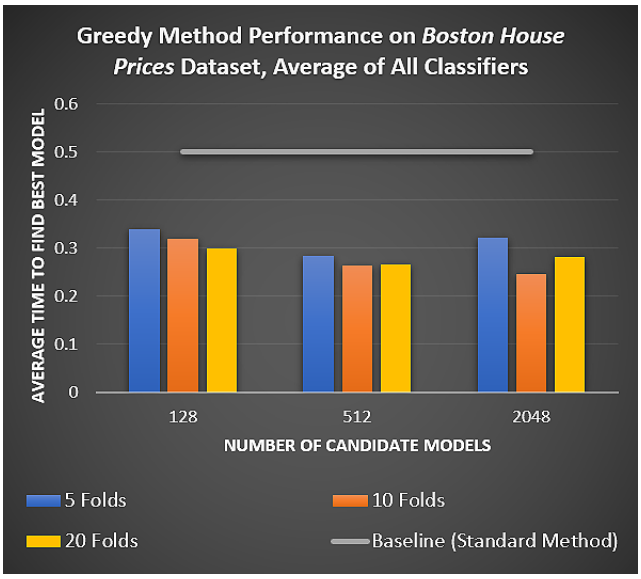

Figure 6. Greedy k-fold method performance for varying numbers of candidate models.

Regardless of the dataset, ML algorithm, number of folds, or number of candidate models, the greedy *k*-fold method was observed to consistently outperform the

standard *k*-fold method on average during the experiments. In the absence of counterevidence, these observations provide support for the notion that the greedy *k*-fold method is generally superior to the standard *k*-fold method in quickly identifying optimal ML models, regardless of the dataset, ML algorithm, number of folds, or number of candidate models. Since the data suggest that the greedy method is, on average, approximately twice as efficient as the standard method in terms of its ability to quickly locate top-performing ML models, it is recommended that the greedy method be given serious consideration in any machine learning hyperparameter tuning / model selection scenario, particularly when an ML practitioner is operating under the common constraint of a computational budget.

6. Summary, limitations, & future research

This paper developed and presented a greedy algorithm for performing *k*-fold cross validation and showed through a large set of experiments that the greedy method clearly and substantially outperforms the standard *k*-fold method in its ability to quickly identify optimal or near-optimal machine learning models. More specifically, given a set of candidate models, the greedy *k*-fold method will, on average, identify the optimal model approximately twice as quickly as the standard method. This means that given a fixed computational budget, approximately twice as many candidate models could be considered by using the greedy *k*-fold method than could otherwise be considered by using the standard method. Alternatively, given a fixed number of candidate models, the greedy *k*-fold method would allow the best-performing models in the set to be identified using just half of the computational budget that would be required to achieve the same results using the standard method. From a practical perspective, these properties of the greedy *k*-fold method can translate to huge savings for companies by reducing the time and money required to develop and train ML-based products and services, thereby yielding substantial gains in competitive advantage. The greedy *k*-fold method can also provide major benefits to AI and machine learning researchers who are developing and performing hyperparameter optimization on complex ML models.

As with all research, this project has several limitations that merit acknowledgement. First, although efforts were taken to test the greedy *k*-fold algorithm on a variety of datasets, those datasets were all relatively small, with the largest dataset containing just 1,797 cases and 64 features. There is some indication among the results presented in Table 1 that the performance of the greedy *k*-fold method may improve on larger datasets (possibly due to less variation among the distributions of each fold), but this notion was not explicitly tested in the current study. Second, while the greedy method was subjected to three different ML algorithms in this project, all of those algorithms were classifiers. There is no obvious *a priori* reason to expect that the greedy *k*-fold method would perform differently for ML algorithms that produce ordinal or continuous predictions. Nevertheless, such algorithms were not used in the current study, which limits the generalizability of the results. Finally, the performance of the greedy *k*-fold method described in the current paper was compared only against the standard *k*-fold method in terms of its ability to quickly identify an optimal model. While the greedy method is unique in terms of its focus on cross validation as a means of accelerating the hyperparameter optimization / ML model selection process, many other approaches to hyperparameter optimization have been proposed, and the greedy *k*-fold method has not yet been compared to those methods.

Ultimately, this paper represents but a small first step in investigating greedy *k*-fold cross validation and its potential as an accelerant for ML hyperparameter optimization and model selection, and much remains to be done. To be sure, the greedy *k*-fold method described in this study is the simplest possible version of the algorithm, and more advanced and better performing algorithms based on the same principles may certainly be feasible. For example, could a more effective approach be developed to handle the exploration / exploitation dilemma? Could the distributional properties of a model's folds be utilized as a basis for early abandonment of unpromising models? Can the greedy *k*-fold method be combined with other approaches designed to accelerate hyperparameter optimization in order to identify optimal ML models even more quickly? All of these questions remain to be answered and hence represent fruitful opportunities for future research in this area. For now, we must content ourselves with the knowledge that the greedy *k*-fold cross validation algorithm appears to be highly promising with respect to its ability to outperform the standard *k*-fold approach, which itself has undeniably been a mainstay of the ML community for several decades.

7. References

[1] Duong, T.N.B., and Sang, N.Q., "Distributed Machine Learning on IAAS Clouds", 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), 2018, pp. 58-62.

[2] Lwakatare, L.E., Raj, A., Crnkovic, I., Bosch, J., and Olsson, H.H., "Large-Scale Machine Learning Systems in Real-World Industrial Settings: A Review of Challenges and Solutions", Information and Software Technology, 127, 2020, pp. 106368.

[3] Feurer, M., and Hutter, F., "Hyperparameter Optimization", in (Hutter, F., Kotthoff, L., and

Vanschoren, J., 'eds.'): Automated Machine Learning: Methods, Systems, Challenges, Springer, Cham, Switzerland, 2019, pp. 3-33.
[4] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A., "A System for Massively Parallel Hyperparameter Tuning", 3rd Machine Learning and Systems Conference, 2020
[5] Snoek, J., Larochelle, H., and Adams, R.P., "Practical Bayesian Optimization of Machine Learning Algorithms", Advances in neural information processing systems, 25, 2012, pp. 2951-2959.
[6] Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.-H., and Patton, R.M., "Optimizing Deep Learning Hyper-Parameters Through an Evolutionary Algorithm", Workshop on Machine Learning in High-Performance Computing Environments, 2015, pp. 1-5.
[7] Li, L., Jamieson, K., Desalvo, G., Rostamizadeh, A., and Talwalkar, A., "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization", The Journal of Machine Learning Research, 18(1), 2017, pp. 6765-6816.
[8] Bengio, Y., "Gradient-Based Optimization of Hyperparameters", Neural computation, 12(8), 2000, pp. 1889-1900.
[9] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M., "Forward and Reverse Gradient-Based Hyperparameter Optimization", 34th International Conference on Machine Learning, 2017, pp. 1165-1173.
[10] Vanwinckelen, G., and Blockeel, H., "Look Before You Leap: Some Insights into Learner Evaluation with Cross-Validation", European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Workshop on Statistically Sound Data Mining, 2014, pp. 3-20.
[11] Kumar, R., Machine Learning Quick Reference: Quick and Essential Machine Learning Hacks for Training Smart Data Models, Packt Publishing, Birmingham, UK, 2019.
[12] Agrawal, T., Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient, Apress, New York, NY, 2020.
[13] Kohavi, R., and John, G.H., "Automatic Parameter Selection by Minimizing Estimated Error", 12th International Conference on Machine Learning, 1995, pp. 304-312.
[14] Olson, R.S., Cava, W.L., Mustahsan, Z., Varik, A., and Moore, J.H., "Data-Driven Advice for Applying Machine Learning to Bioinformatics Problems", Pacific Symposium on Biocomputing, 2018, pp. 192-203.
[15] Bergstra, J., and Bengio, Y., "Random Search for Hyper-Parameter Optimization", Journal of machine learning research, 13(1), 2012, pp. 281-305.
[16] Soper, D.S., "On the Need for Random Baseline Comparisons in Metaheuristic Search", 51st Hawaii International Conference on System Sciences, 2018, pp. 1288-1297.
[17] Brownlee, J., Probability for Machine Learning, Machine Learning Mastery Pty. Ltd., Vermont, Australia, 2019.
[18] Iba, H., Evolutionary Approach to Machine Learning and Deep Neural Networks, Springer, Gateway East, Singapore, 2018.
[19] Jamieson, K., and Talwalkar, A., "Non-Stochastic Best Arm Identification and Hyperparameter Optimization", 8th International Conference on Artificial Intelligence and Statistics, 2016, pp. 240-248.
[20] Karnin, Z., Koren, T., and Somekh, O., "Almost Optimal Exploration in Multi-Armed Bandits", 30th International Conference on Machine Learning, 2013, pp. 1238-1246.
[21] Larsen, J., Hansen, L.K., Svarer, C., and Ohlsson, M., "Design and Regularization of Neural Networks: The Optimal Use of a Validation Set", Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop, 1996, pp. 62-71.
[22] Maclaurin, D., Duvenaud, D., and Adams, R., "Gradient-Based Hyperparameter Optimization Through Reversible Learning", 32nd International Conference on Machine Learning, 2015, pp. 2113-2122.
[23] Pedregosa, F., "Hyperparameter Optimization with Approximate Gradient", 33rd International Conference on Machine Learning, 2016, pp. 737-746.
[24] Duchi, J., Hazan, E., and Singer, Y., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", Journal of machine learning research, 12, 2011, pp. 2121-2159.
[25] Kingma, D.P., and Ba, J.L., "Adam: A Method for Stochastic Optimization", 3rd International Conference on Learning Representations, 2015
[26] Shalev-Shwartz, S., and Ben-David, S., Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, Cambridge, UK, 2014.
[27] Allen, D.M., "The Relationship Between Variable Selection and Data Agumentation and a Method for Prediction", technometrics, 16(1), 1974, pp. 125-127.
[28] Stone, M., "Cross-Validatory Choice and Assessment of Statistical Predictions", Journal of the Royal Statistical Society: Series B (Methodological), 36(2), 1974, pp. 111-133.
[29] Cawley, G.C., and Talbot, N.L., "On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation", The Journal of Machine Learning Research, 11, 2010, pp. 2079-2107.
[30] Das, S., and Cakmak, U.M., Hands-On Automated Machine Learning, Packt Publishing Ltd., Birmingham, UK, 2018.
[31] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., and Dubourg, V., "Scikit-Learn: Machine Learning in Python", Journal of machine learning research, 12, 2011, pp. 2825-2830.
[32] Welch, B.L., "The Generalization of "Student's" Problem When Several Different Population Variances are Involved", Biometrika, 34(1-2), 1947, pp. 28-35.