CS-E4870 - Research Project in Machine Learning and Data Science

Cross validation pruning using Bayesian methods

January 2, 2020

Emil Kreutzman 425902 emil.kreutzman@aalto.fi

Abstract

I present a Bayesian method to model mean validation accuracy values computed in the process of performing hyperparameter optimization when training a machine learning model. The Bayesian model is based on the Beta distribution. Further, I formulate an algorithm by which said statistical model can be exploited to perform cross validation faster, by pruning CV iterations for hyperparameters that are likely to yield poor performance.

1 Introduction

In machine learning applications, a common way to optimize the output function in the hyperparameter space, is cross validation (CV). The objective is to find the set of hyperparameters (for said ML algorithm) that generalizes the best; that is, fits well to previously unseen data outside of the training set.

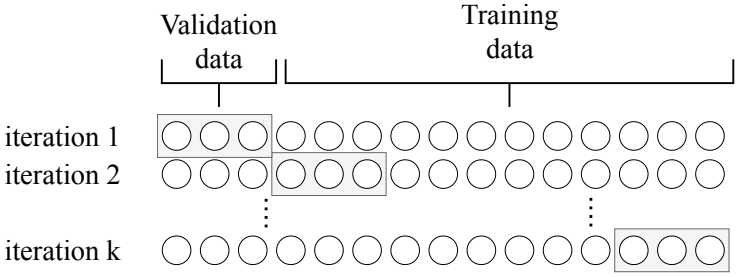

Figure 1: k-fold cross validation.

While there are other approaches too, a common way to perform cross validation is to divide the training data into k folds. Successively, each fold is then used as validation data while the other $k-1$ folds are used to train the machine learning model. As such, the computational complexity of the optimization process grows according to

$$\mathcal{O}(N\,k),$$

where N is the number of hyperparameter combinations to be explored and k is the number folds (and thus, iterations) used for each set of hyperparameters.

While numerous Bayesian optimization methods [1] have been proposed to efficiently explore the hyperparameter space (reduce effective N), none focus on reducing k. This is the problem in cross validation: we too often diligently evaluate[1] k rounds of cross validation even if the first folds show poor results. A speedup can be achieved if we can assess the quality of hyperparameters without evaluating all k folds. This is especially true for larger machine learning models, where training the model once is very computationally costly.

The method proposed in this paper does not aim to replace well-known Bayesian optimization methods ([1]) for exploring the hyperparameter space, but rather serve as a complement and reducing the effective number of cross validation iterations k needed to evaluate each set of hyperparameters. As such, my method solves a different problem altogether even if I also use Bayesian methods. More on this in section 4.

[1] There are actually many methods (eg. [2]) to prune cross validation iterations, but they are heuristic in nature and lack proper statistical rigidity.

2 Bayesian pruning of cross validation folds

I present a Bayesian model for predicting the mean validation accuracy for one set of hyperparameters of a machine learning model.

2.1 Beta model of validation accuracy values

In performing k-fold cross validation for a classification task, one can calculate an accuracy value $a_{i,j}$ for each iteration among the k validations. For a model on hyperparameter set j, this can be expressed as:

$$a_{i,j} \in [0, 1],\ i \in [1, ..., k] \tag{1}$$

It is convenient to assume that the accuracy values are Beta-distributed (Eq. 2) for the following reasons:

- All accuracy values fall in the $[0, 1]$ interval. This is also the support of the Beta distribution.
- Unless both parameters α and β fall in the $(0, 1]$ range, the corresponding Beta distribution has a single mode. This captures our belief that validation accuracy values in training a machine learning model are somewhat concentrated around some mean accuracy.
- There are lots of random factors that contribute to the innate randomness of training a machine learning model. This includes random initialization of model parameters, as well as different stopping criteria in a single iteration of training. This randomness doesn't necessarily call for a Beta model, but somewhat justifies why any cost function of a machine learning model will yield randomness in training. For accuracy specifically, we can model this using a Beta.

$$a_1, a_2, ..., a_k \sim \text{Beta}(\alpha, \beta) \tag{2}$$

To perform inference using the Beta likelihood, we'll reparametrize and choose sufficient prior distributions. Eq. 3 shows the new parameters of the Beta distribution based on the model parameters α and β. Since α, β are subject to constraints $\alpha > 0$ and $\beta > 0$, it follows that $0 < \eta < 1$ and $\mu > 0$.

We reparametrize according to

$$\begin{aligned} \alpha &= \mu \cdot \eta \\ \beta &= \mu \cdot (1 - \eta), \end{aligned} \tag{3}$$

and define the model to be

$$\begin{aligned} \mu &\sim \text{Exp}(10^{-2}) \\ \eta &\sim \text{Beta}(1, 1) \\ a_1, a_2, ..., a_k &\sim \text{Beta}(\alpha, \beta). \end{aligned} \tag{4}$$

Note that while the Beta distribution does have a family of conjugate priors they are intractable [3]. As such, we're better off choosing priors that match our beliefs and can be simulated numerically in Bayesian modeling software.

2.2 The prior choices

The full model defined in Eq. 4 has some peculiar choices that I will now explain. We will also see examples of the posterior predictive probability densities, fit on hypothetical datasets.

2.2.1 The μ and η parameters

Reparametrizing into μ and η helps us think of the model in a more intuitive way.

Different prior distributions on μ will affect how large (scale) α and β likely will be in the posterior. The scale is exponential, which makes the Exp distribution a suitable prior on μ. From the reparametrization follows that $\alpha + \beta = \mu \cdot \eta + \mu \cdot (1 - \eta) = \mu$, and as such the prior on μ allows us to set a distribution on how large we think $\alpha + \beta$ could grow.

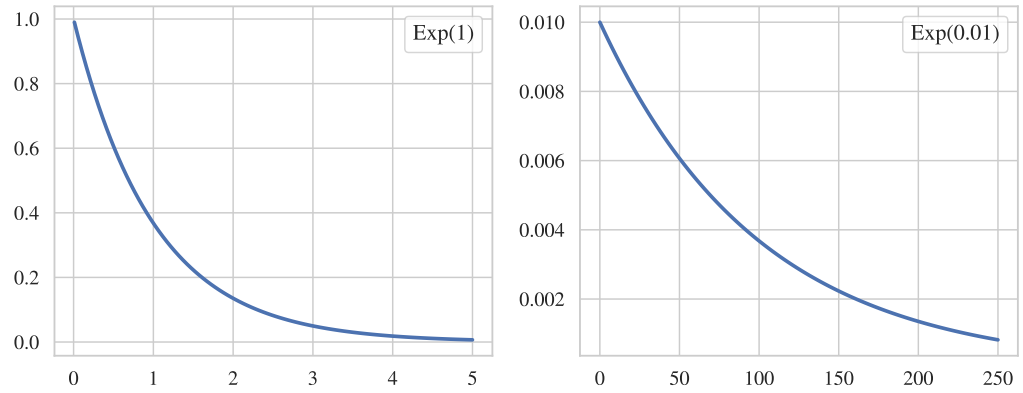

Figure 2: Exponential distribution. Comparison of fallof using different parameters for the exponential prior on μ.

Fig. 2 shows how the model's exponential prior $\text{Exp}(10^{-2})$ compares to the distribution $\text{Exp}(1)$. For $\text{Exp}(1)$ 99.3% of the mass exists in $(0, 5]$; it wouldn't be a great as a prior choice on μ since it would be stating a belief of *only* small scales on μ and by extension, $\alpha + \beta$. Rather, a sensible choice is $\text{Exp}(10^{-2})$ since it captures the belief that the scale is likely somewhat small, but could also be in the hundreds. Finally, note that the prior on μ does not have to be the exponential distribution, necessarily. It could also be Gamma, or some other distribution that defines exponentially decaying densities in $(0, \infty)$.

The η parameter defines roughly the *skew* of the model, similarly to how μ defines the scale. For instance, a value $\eta = 0.1$ will skew the posterior towards β, since $\beta = \mu \cdot (1 - 0.1) > \alpha = \mu \cdot 0.1$.

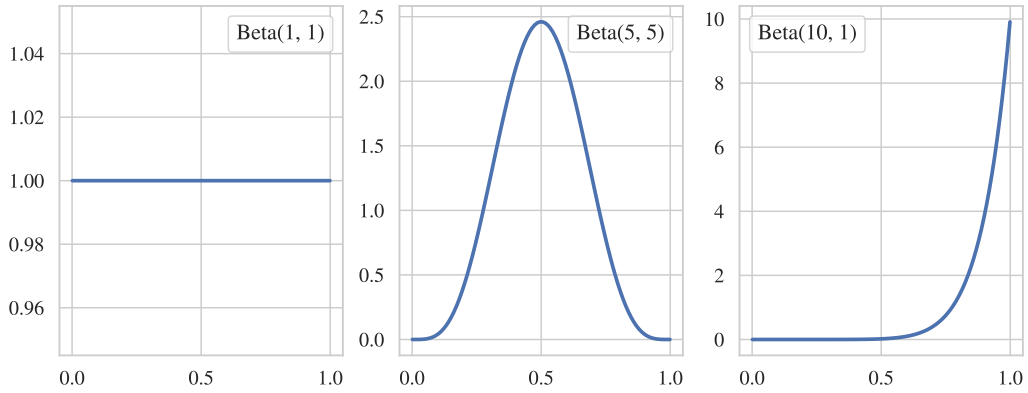

Figure 3: Different Beta distributions. The Beta is a sensible prior choice for η defining the skew of the model because of the [0, 1] support.

Different Beta distributions are shown in Fig. 3. In the end, the uniform $\text{Beta}(1, 1)$ prior on η is the best choice since there really isn't anything we can say about the ratio of α and β in the model, before accuracy values are observed.

2.2.2 Prior effects on posterior

In the last section 2.2.1 I discussed how and why the prior choices on μ and η make sense in the model. Now, let us look at how the model choices made affect the posterior and prior predictive distributions.

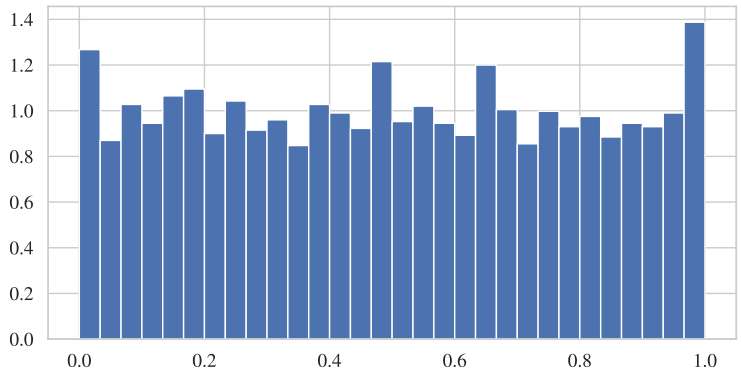

Figure 4: The prior predictive distribution. It is close to perfectly uniform; quite suitable for our model of ML model accuracy distributions.

The prior predictive distribution of the model is shown in Fig. 4. Notice how the distribution is slightly concave, with larger densities close to the interval edges. This is expected, since the Beta distribution is concave when both of its parameters α and β are in the $(0, 1)$ interval, and our prior $\mu = \alpha + \beta$ *is defined* in that range. We could rectify this non-uniformity by adjusting the model, but such measures would not contribute to additional usefulness in applying the model, while still making it more complex. Additionally, even the current prior predictive is essentially uniform for all practical purposes, so this will not be an issue.

Next, I show what the posterior predictive distribution looks like for our model using different simulated datasets A. The results of the simulation are shown in Fig. 5. There isn't anything particularly notable about the distributions; they resemble our expectations for a good Bayesian model applicable to the problem at hand.

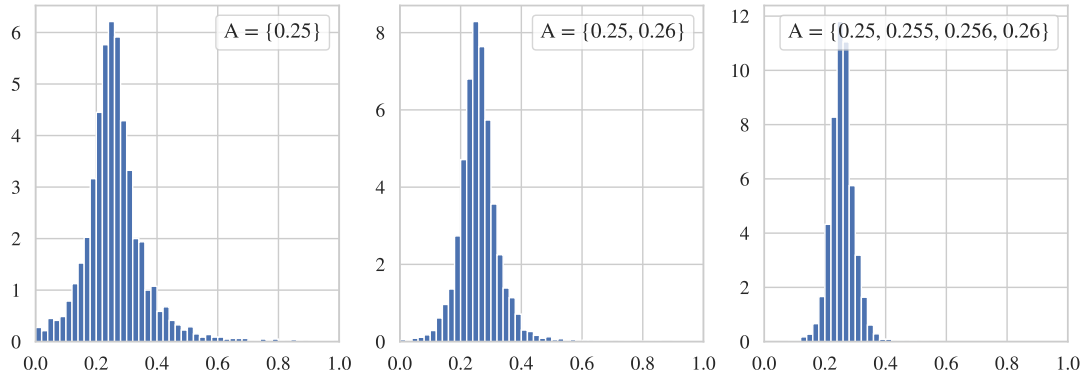

Figure 5: The posterior predictive distribution given different simulated datasets A, with mean accuracy values.

Finally, in Fig. 6 I contrast the Bayesian model to a naive maximum likelihood (ML) fit on corresponding data. One could perhaps argue that the simpler ML fit for the parameters of Beta(α, β) should be sufficient for a utility model such as the one in this paper. However, the figure shows clearly that a dataset $A = \{0.80, 0.82\}$ with small variance causes a very sharp ML fit, thus likely underestimating the posterior variance. The Bayesian model (Post. pred.) shows higher variance, in line with our expectations for a dataset of only 2 observations. On a dataset with observations $A = \{0.4, 0.5\}$ of higher variance, the two model densities align more closely.

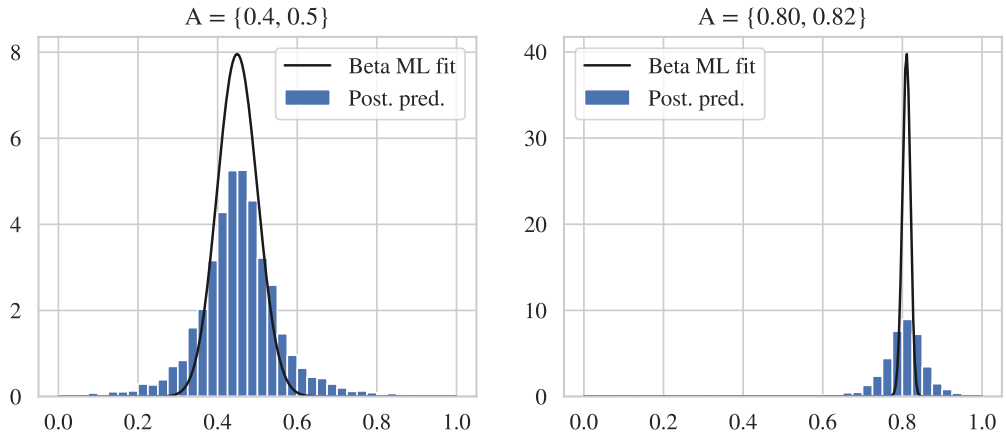

Figure 6: While maximum likelihood estimates are similar to the Bayesian approach for datasets with greater variance (left), the ML approach overfits severely when the variance in A is small (right).

3 Pruning algorithm

I propose an algorithm for pruning CV iterations in a statistically valid way, using the Bayesian model from Eq. 4.

3.1 Pruning criterion

In the previous section I defined a Bayesian Beta model that can be used to model validation accuracy values retrieved in training a machine learning model for 1 set of hyperparameters (many CV iterations). In itself however, this is not useful if we cannot compare distributions from different machine learning models (hyperparameters), and prune CV iterations for models deemed bad performers.

Given two sets of accuracy values $A_1 = \{a_{11}, a_{12}, ..., a_{1m}\}$ and $A_2 = \{a_{21}, a_{22}, ..., a_{2n}\}$ from two sets of hyperparameters H_1 and H_2 (2 ML models), we may fit the Bayesian Beta model from Eq. 4 on each dataset of accuracy values. Thus, we would have two posterior predictive fits M_1 and M_2. If said model densities correspond to random variables X_1

and X_2, respectively, we may compare them in the following manner:

$$P(X_1 > X_2) = \int \int^{x} f(x = X_1)\, f(y = X_2)\, dy\, dx \tag{5}$$

Now, if $P(X_1 > X_2) > \tau$ under some confidence threshold τ (eg. 0.99), we may be certain that hyperparameters H_2 will not yield better generalization than H_1. This will be our pruning criterion to remove H_2 from further consideration, and not evaluate further CV iterations at all. When we make this decision to not evaluate H_2 further, we save on computing time for the remaining, skipped CV iterations.

Additionally, note that the previous pruning statement is true regardless of how many items are in the sets A_1 and A_2 being compared. We may evaluate only 1 item for multiple hyperparameter sets and still perform pruning of poor results among those sets evaluated.

I now propose an algorithm to use the Beta model of accuracies, as well as the pruning criterion defined in Eq. 5.

Algorithm 1 Cross validation pruning using Beta model

procedure TRAIN($b = 10, \tau = 0.99$) ▷ b is the buffer size, posteriors in memory.
  $buffer_h \leftarrow \varnothing$ ▷ List of hyperparameters for the ML model.
  $buffer_a \leftarrow \varnothing$ ▷ List of mean accuracy values for each model.
  $buffer_p \leftarrow \varnothing$ ▷ List of posteriors Beta(α, β).
  while True **do**
    if $|buffer_b| < b$ **then**
      $h \leftarrow$ next hyperparameters
      $buffer_h \leftarrow buffer_h \cup \{h\}$
      $buffer_a \leftarrow buffer_a \cup \varnothing$ ▷ Each value in $buffer_a$ is an array.
      $buffer_p \leftarrow buffer_p \cup null$
    $remain \leftarrow \{i \in [1, ..., k] \mid |buffer_a^i| < k\}$
    sort $remain$ by $mean(a_i)$ for $i \in remain$, highest first
    $j \leftarrow remain_1$
    $a \leftarrow train\_and\_validate(buffer_h^j)$ ▷ Train ML model, calculate accuracy
    $buffer_a^j \leftarrow buffer_a^j \cup a$
    $buffer_p^j \leftarrow$fit $Beta$ with data $buffer_a^j$ ▷ Store posterior distribution.
    $i_{best} = argmax(mean(a_i) \in buffer_a^i| < k)$
    if end_condition **then** ▷ Choose your own end condition.
      return $buffer_h^{ibest}$ ▷ Return best hyperparameters.
    for $i = 0$; $i \leq |buffer_p|$; i++ **do**
      if $P(buffer_p^{ibest} > buffer_p^i) > \tau$ **then** ▷ This is where we prune.
        delete $buffer_h^i, buffer_a^i, buffer_p^i$

Alg. 1 exploits the fact that we do not need to evaluate all cross validation iterations for *any* set of hyperparameters, but can make valid comparisons even if we only did 1 or 2 CV iterations for some ML models. It would also be valid to perform regular cross validation, and just keep the previous best (highest mean of posterior predictive) model as the reference.

See the Github page for this paper, for a full reference implementation of Alg. 1.

3.2 Experiments and benchmarking

While the main thesis in this paper has not been showing that pruning cross validation iterations can speed up computation time (evidently [2], this is possible), I show that this method can be used for the same purpose.

The reference implementation shows the algorithm presented in the previous section, applied on training a neural network for classifying hand-written digits in the MNIST dataset. Today the problem itself is trivial, but it can still be shown how Bayesian pruning of CV validation can be applied to reduce computation time. In Fig. 7 I compare a run using my algorithm in the previous section, to regular k-fold cross validation.

In the figure, we can see how the method (red) is rapidly able to explore far more of the search space in the beginning of the run, because of the bufferized models. With 20 trained models, the Bayesian pruning approach has already evaluated 19 hyperparameter sets (19 different ML models), compared to only 4 for the regular 5-fold cross validation. The yield is not as big after some number of machine learning models are put into the buffer.

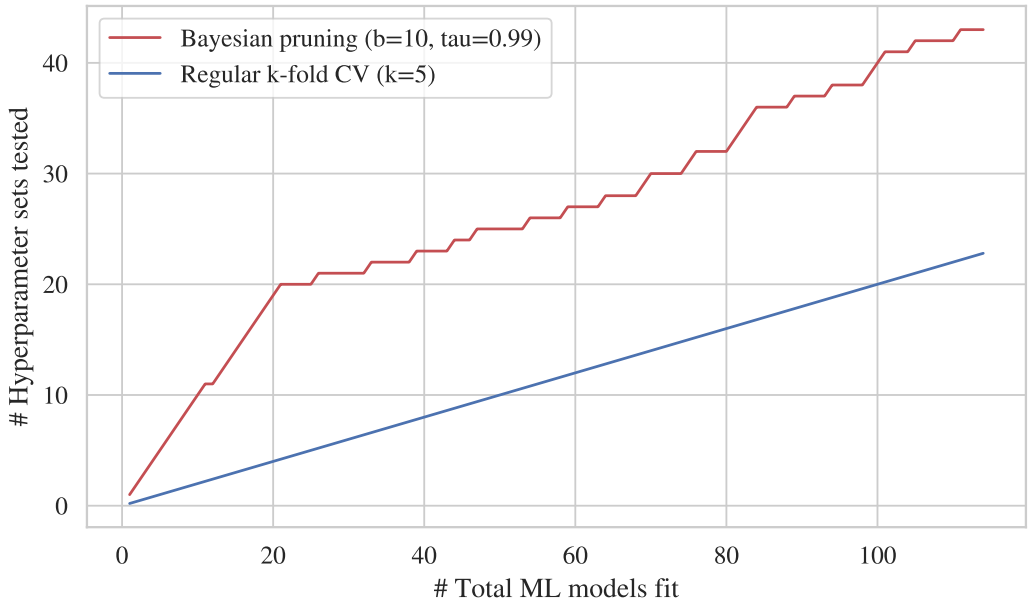

Figure 7: Example benchmark showing the benefit of using the Bayesian CV pruning versus regular k-fold CV.

4 Related work

While this paper presents a novel approach for performing cross validation efficiently, previous work within hyperparameter optimization has often focused on exploring the hyperparameter space itself.

In regular grid search of course, the architect of the machine learning model defines a range of possible parameter combinations in a multidimensional *grid*. Each combination of parameters is then evaluated against the objective function, and the best set is chosen in the end. For machine learning specifically, it has been shown that random sampling of hyperparameters is more efficient than regular grid search. Bergstra and Bengio study random search and grid search in the same parameter domains to show how random search yields better model performance and saves on computing time too. [4] Perhaps surprisingly, random search in the hyperparameter space sometimes even outperforms more advanced Bayesian optimization approaches to finding good hyperparameters; but not always. A popular choice for Bayesian optimization is to model the objective function (generalization performance of the ML model) using Gaussian processes. Using this approach, we evaluate each set of hyperparameters and assume the validation accuracy is a sample from some random unknown function. The goal becomes finding the point in the hyperparameter space that maximixes this objective. The Gaussian process is used to approximate this function, and we may perhaps hone in on a point in the hyperparameter space that maximizes the objective faster (fewer parameter samples) than we otherwise would. [5]

It is noteworthy that the aforementioned methods are distinctly different to the methods presented in this paper, and they're not mutually exclusive as cross validation can be used with both random search and Bayesian optimization in the hyperparameter space. A more similar method to mine is presented by Gabryś, who refers to their approach simply as "pruned cross validation". [2] In that method, the blog post instructs model authors to "pick a value $t > 0$, eg. 0.1" and prune iterations if the mean score (eg. mean accuracy) multiplied by $1 + t$ is still below the mean score of the best round so far. Crucially, this method doesn't employ any statistics, and as such, is necessarily just heuristic.

5 Conclusion and future work

In this paper I showed how Bayesian methods can successfully be used to prune CV iterations that will not yield good generalization in terms of classification accuracy. When training a machine learning model, this allows for selecting a wider hyperparameter search

space, faster hyperparameter optimization overall and better utilization of available computing resources. Unlike other similar methods, this new method introduces a statistical model for making pruning decisions to avoid making poor decisions because of high uncertainty.

Additionally, I introduced an algorithm to take advantage of the fact that the presented Bayesian model can be used to model the objective function even after very few samples. Using this algorithm is not necessary for using the Bayesian model in pruning CV iterations, but it could make the hyperparameter optimization more efficient. On a toy problem using the MNIST dataset of handwritten digits, I conclude that the algorithm can indeed speed up cross validation by pruning unnecessary iterations. However, further analysis on the speedups is still necessary.

There is still plenty of work to be done within the domain of speeding up the training process of machine learning models. A major drawback of the methods presented in this paper, is that the only objective function measured is validation accuracy. It would be useful to define similar Bayesian models for other objectives, such as *log-loss*. I also leave to future papers, the effort of defining the pruning process more rigorously in terms of Bayesian decision analysis.

6 Appendix

All the code used in this paper, along with a reference implementation of the pruning algorithm presented: https://github.com/ekreutz/bayes-cv-pruning.

References

[1] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[2] "Pruned cross validation for hyperparameter optimization." https://piotrekga.github.io/Pruned-Cross-Validation/, 2019. Accessed: 2019-11-02.

[3] C. P. Robert and G. Casella, *Introducing Monte Carlo Methods with R (Use R)*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2009.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[5] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.