Efficient Machine Learning Hyperparameter Tuning Using CMA-ES

Rebecca McBrayer

Northeastern University
mcbrayer.r@northeastern.edu

April 29, 2021

Abstract

The challenge of efficient hyperparameter tuning is one with many real-world consequences in data science. Model accuracy, training time, and environmental impact are all important considerations when determining what model to use and what hyperparameters to configure said model with. A grid search method checking all possibilities from a discrete number of parameters is used most commonly, but this is ineficcient and has a number of downsides. In this paper I show that evolutionary algorithms, namely the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES), is much better suited to the task of hyperparameter tuning than grid search. Specifically, I show that CMA-ES can be used to provide minor improvements to the best cross-validation accuracy score while cutting search times by almost 50%. Code at: `https://github.com/eccabay/CMA-ES_hyperparameters/tree/master`

I. Introduction

All machine learning models, whether traditional or deep learning, supervised or unsupervised, require a variety of hyperparameters to be selected prior to training. The values of these hyperparameters can have a significant impact on the performance of the model, but there is no systematic way to determine the best values for any given problem. Over the past 30 years, some research has explored different methods for hyperparameter tuning, ranging from grid search to Bayesian optimization to evolutionary strategies. These studies have shown that Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is a high performing optimization strategy for this sort of task [10]. However, the research in this field has focused primarily on parameter optimization for SVMs and deep learning techniques [11]. While deep neural networks are currently state-of-the-art for the large majority of machine learning tasks, the traditional machine learning methods are still in common use thanks to their simplicity, faster training times, and interpretability. The performance of CMA-ES in hyperparameter tuning for traditional machine learning models is therefore an important research topic, but one that has not been thoroughly explored.

The goal of this project is to apply the CMA-ES algorithm to traditional machine learning models within the Automated Machine Learning (AutoML) space for supervised classification tasks. Using a collection of popular models, I use CMA-ES to select the best model with the best hyperparameters for a small number of example datasets. The goal is to maximize model cross-validated accuracy while minimizing training time, as well as overall hyperparameter search time. To compare against a baseline, I run a grid search with a discrete set of hyperparameter values for each model.

Table 1: *Testing Datasets*

Dataset Type	Dataset Name
Binary Classification	Heart Attack Analysis and Prediction [2]
	Stroke Prediction [3]
	Telecom Users [4]
Multiclass Classification	MNIST [1]
	Forest Covertypes [1]
	Kepler Exoplanet Search Results [5]
Regression	California Housing [1]
	Melbourne Housing [6]
	World Happiness Report 2021 [7]

II. Data

Different subsets of machine learning problems have different optimal hyperparameters. To determine generalizability of these methods, I run this hyperparameter testing on a variety of datasets. I have three unique datasets for each distinct supervised machine learning method: binary and multiclass classification, along with regression. These datasets can be seen in Table 1. All features of these datasets are either numerical or categorical, so there is no need for any natural language processing methods in order to prepare the data.

I search for the best hyperparameters using a specific subset of scikit-learn implementations [1] of commonly used estimators on each of the listed datasets. Since classification and regression are distinct tasks, I use Logistic Regression for the classification datasets and Linear Regression for the regression datasets. Other than that, however, all estimators are the same baseline concept: K Nearest Neighbors, Decision Trees, and AdaBoost. This set of estimators was chosen to span as many different types of prediction as possible with the fewest distinct models.

III. Methods

To enable easy training on the nine datasets previously mentioned, I have written a series of data loading and preprocessing functions. These functions load each dataset, impute any missing values to be the most frequent class, and convert any text or otherwise non-integer columns to numerically categorical columns instead. None of the datasets being used for this project contain any features that could be considered under the umbrella of natural language processing, they all can be converted to categorical columns without the loss of any information.

The baseline Grid Search implementation uses scikit-learn's GridSearchCV class, which checks the performance of a pre-set list of hyperparameter options on a given estimator. These were chosen relatively arbitrarily, simply a few values around the defaults given by the scikit-learn documentation and on the same order of magnitude. The algorithm checks each possible combination of these predetermined parameters and reports the set that gives the highest accuracy from 3-fold cross-validation at the end, brute forcing a solution from a discrete set of possibilities.

An existing Python implementation of CMA-ES lives on PyPI [8], but this work adapts that code base in order to fit the requirements of the hyperparameter tuning task. This CMA-ES implementation is written as a class which parallels the GridSearchCV implementation, searching for the best hyperparameters for a single estimator and dataset pair at a time.

Table 2: *Overall Results*

	Grid Search			CMA-ES		
	Best Classifier	Best Score	Time Taken	Best Classifier	Best Score	Time Taken
Heart Attack	Logistic Reg.	0.831	123 sec	Logistic Reg.	0.827	33 sec
Stroke	Logistic Reg.	0.955	234 sec	Logistic Reg.	0.955	37 sec
Telecom	Logistic Reg.	0.800	298 sec	Logistic Reg.	0.798	70 sec
California	Decision Tree	0.628	397 sec	Decision Tree	0.667	286 sec
Melbourne	Decision Tree	0.628	833 sec	Decision Tree	0.657	173 sec
Happiness	AdaBoost	0.774	56 sec	AdaBoost	0.790	30 sec
MNIST	K Neighbors	0.985	632 sec	K Neighbors	0.981	142 sec
Forest	K Neighbors	0.803	3409 sec	K Neighbors	0.783	2168 sec
Kepler	AdaBoost	0.985	992 sec	AdaBoost	0.987	817 sec
Average		0.821	775 sec		0.827	417 sec

The CMA-ES search starts from a set of default hyperparameters, which are directly pulled from the scikit-learn documentation. From there, a series of similar hyperparameters are proposed by sampling from a normal multivariate distribution, and then each is evaluated based on accuracy score of the model trained using these parameters, from 3-fold cross-validation. The mean and covariance matrix of the normal distribution that new hyperparameters are sampled from are then updated using the best-performing parameters from the previously sampled generation. This process is repeated until either the maximum number of iterations is reached or proposing new parameters no longer produces an improvement in accuracy score [9]. This approach allows for searching parameters that are skipped over in the Grid Search method, to spend more time focusing on hyperparameters that perform better on the given dataset.

Both the grid search and CMA-ES classes are wrapped in respective search classes which handle dataset iteration and reporting of results. I run the same set of experiments with both of these hyperparameter searching applications. Using estimators Logistic/Linear Regression, KNN, Decision Trees, and AdaBoost on the set of nine datasets from scikit-learn and Kaggle, I run the search algorithms over each dataset and report the best estimator for each, with the top-performing parameters and resulting score. I also track the total amount of time taken, for each dataset and for each prediction type overall.

IV. Results

As can be seen in Table 2, CMA-ES provides significant improvement over the naive grid search method. The Best Score columns are the training score from 3-fold cross validation, and there is a slight improvement from an average score of 0.821 to 0.827. This improvement also exists in the testing scores, from 0.828 with grid search to 0.835 with CMA-ES. The most significant speedup, however, comes with the amount of time taken, reducing the average run time of 775 seconds down to 417. The most significant time improvement can be seen on the Melbourne Housing dataset (833 seconds to 173 seconds) and the MNIST dataset (632 seconds to 142 seconds).

Since the time improvement is the most significant improvement CMA-ES provides, it becomes interesting to examine how the search timing varies between the different estimators. We take a look at the datasets with the largest improvement with CMA-ES from each type of search problem (binary and multiclass classification, and regression), namely the Stroke dataset, the

Table 3: *Per-Estimator Time Taken on Stroke Dataset*

		Logistic Reg.	KNN	Decision Tree	AdaBoost
Stroke	Grid Search	135 sec	44 sec	3 sec	52 sec
	CMA-ES	6 sec	6 sec	3 sec	22 sec
Melbourne	Grid Search	1 sec	442 sec	23 sec	368 sec
	CMA-ES	1 sec	50 sec	24 sec	99 sec
MNIST	Grid Search	515 sec	31 sec	6 sec	79 sec
	CMA-ES	46 sec	1 sec	3 sec	93 sec

Table 4: *Parameter Comparison*

		Max Depth	Min Samples Split
California	Grid Search	225	5
	CMA-ES	165	20
Melbourne	Grid Search	25	5
	CMA-ES	160	0.0148

Melbourne Housing dataset, and the MNIST dataset. From 234 seconds down to 37 seconds, CMA-ES took only 15.7% of the time that grid search took with Stroke. With almost as much improvement, the Melbourne Housing dataset took 20.7% of the time with CMA-ES compared to grid search, and MNIST took 22.5%. These comparisons can be seen in Table 3. Here, we see that fitting the Decision Tree estimators did not provide significant improvement between the two hyperparameter search methods. Logistic Regression search time was reduced by an average of 62%, AdaBoost by 37%. The time it took to search the hyperparameters for K Nearest Neighbors was reduced by a whopping 91%. This makes some sense, as Logistic Regression and K Nearest Neighbors had the largest number of predefined hyperparameters to search: Logistic Regression had 216, Linear Regression had 2, K Nearest Neighbors had 154, Decision Trees had 40, and AdaBoost searched through 66.

Some of the estimators do show accuracy improvement from grid search to CMA-ES. The two datasets that showed the highest accuracy improvement when using CMA-ES instead of grid search were the two housing datasets, California and Melbourne, whose best-performing parameters can be seen in Table 4. Their best estimators were both Decision Tree Regressors, both in the grid search run and in the CMA-ES run. Note that the Decision Tree estimators had the least improvement in search time, as discussed in the previous paragraph. The reason behind both of these results is probably due to the relatively small number of hyperparameter options that were predefined for the grid search. Since scikit-learn recommends starting with a *min_samples_split* of 2 and a *max_depth* of None, the grid search considered *min_samples_split* values of 2, 3, 4, 5 and *max_depth* values from 25 to 250 at intervals of 25. The key difference in this case, then, is the *min_samples_split* value. With the California dataset, the best value was all the way at 20, which was much higher than the grid search even considered. The float value for *min_samples_split* with the Melbourne housing represents the proportion of the total samples passed in during training. Specifically, $min\_samples\_split = ceil(min\_samples\_split * n\_samples)$. The Melbourne training data has 13851 rows, and with 3-fold cross validation each *n_samples* is 9054, meaning the *min_samples_split* on each fold is 137. This value is, once again, much higher than the maximum

value of 5 searched during grid search.

Overall, it is clear that CMA-ES is a much more efficient way to search for hyperparameters than a brute-force grid search. The average time taken to search for hyperparameters was reduced by 46%, almost doubling the speed at which good hyperparameters can be found for a given model on a given dataset. In the few cases where there was no significant improvement in search time, this was offset by a non-trival improvement in model performance by searching values that the grid search could not. With all this in mind, CMA-ES provides two main improvements over grid search: dramatically increased search speed, and the ability to search any and all possible hyperparameter values, rather than being limited by pre-set values as in grid search.

References

[1] Pedregosa et al. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[2] Rahman, Rashik. (2021, March). Heart Attach Analysis & Prediction Dataset, Version 2. Retrieved 05 April, 2021 from `https://www.kaggle.com/rashikrahmanpritom/heart-attack-analysis-prediction-dataset`

[3] Fedesoriano. (2021, January). Stroke Prediction Dataset, Version 1. Retrieved 05 April, 2021 from `https://www.kaggle.com/fedesoriano/stroke-prediction-dataset`

[4] Zosimov, Radmir. (2021, February). Telecom users dataset, Version 1. Retrieved 05 April, 2021 from `https://www.kaggle.com/radmirzosimov/telecom-users-dataset`

[5] NASA. (2017, October). Kepler Exoplanet Search Results, Version 2. Retrieved 05 April, 2021 from `https://www.kaggle.com/nasa/kepler-exoplanet-search-results`

[6] DanB. (2018, June). Melbourne Housing Snapshot, Version 5. Retrieved 05 April, 2021 from `https://www.kaggle.com/dansbecker/melbourne-housing-snapshot`

[7] Singh, Ajaypal. (2021, March). World Happiness Report 2021, Version 2. Retrieved 05 April, 2021 from `https://www.kaggle.com/ajaypalsinghlo/world-happiness-report-2021`

[8] Shibata, Masashi. (2021, February). Lightweight Covariance Matrix Adaptation Evolution Strategy implementation. Retrieved 05 April, 2021 from `https://pypi.org/project/cmaes/`

[9] Hansen, Nikolaus. (2016, April). The CMA Evolution Strategy: A Tutorial. arXiv:1604.00772 .

[10] Hutter, F, Kotthoff, L, and Vanschoren, J. (2019). Automated Machine Learning: Methods, Systems, Challenges. *Springer*, p.3-33.

[11] Loshchilov, I, and Hutter, F. (2016, April). CMA-ES for Hyperparameter Optimization of Deep Neural Networks. arXiv:1604.07269.