Reducing Hyperparameter Search Space of ML models

Akshay Jadiya, Saurabh Aggarwal

January 6, 2022

1 Introduction

Most Machine Learning (ML) algorithms optimize an underlying objective function to fit the training data. The output of the optimization process is a set of parameters that minimize or maximize the objective function. However, some ML algorithms have some parameters - known as hyperparameters - which are not a part of optimization process and are supposed to be adjusted by the user to obtain the best fitting model. A couple of the most common techniques to decide hyperparameter values are Grid Search (which is a brute force approach to try all values in pre-defined hyperparameter search space) and Random Search (which randomly chooses values out of a predefined hyperparameter search space). Both of these approaches select the hyperparameter values that maximize the metric of interest (accuracy , F1 score etc.). In this project, a novel method to select hyperparameters based on experimental design is presented which is much faster than Grid Search and can give better insights into the model as compared to Random Search. The method uses Orthogonal Arrays to reduce the search space and MaxPro designs to select the range of hyperparameter values on which the models will be trained and tested. In this project, the task of digit classification using MNIST dataset is performed using Decision Trees. Comparison of model accuracy and time taken for hyperparameter tuning has been done for Random Search, Grid Search and the proposed OA/MaxPro based search.

2 Background - Machine Learning

We have used Scikit-learn's implementation of Decision Tree classifier [7] to classify the MNIST dataset. Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piece-wise constant approximation. The general algorithm to create a decision tree consists of the following broad steps:

1. Select Root node (based on Gini Index and Highest Information Gain)
2. Pick the best attribute/feature based on the splitting criterion (the best attribute is one which best splits or separates the data)
3. Split the set to produce the subsets of data, i.e., branches in a tree
4. Go to step 2 until full tree is created (or some stopping condition is met)

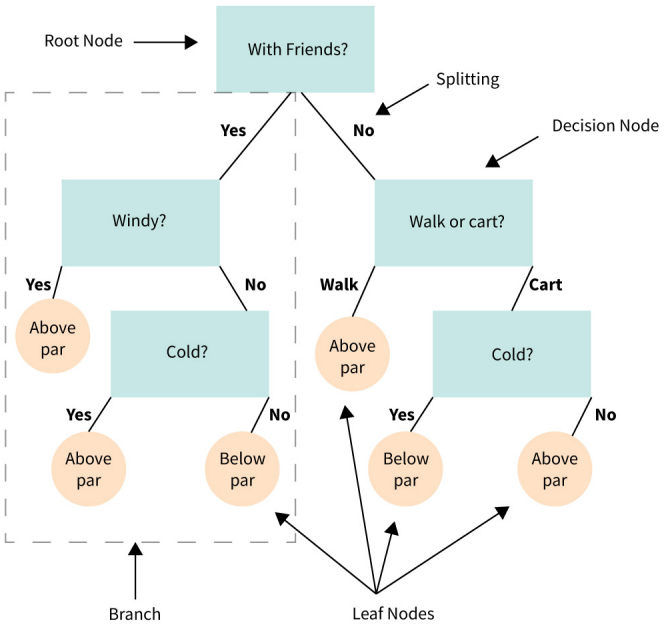

Figure 1: Decision Tree Illustration [3]

Although there are several hyperparameters that can affect the performance of a decision tree, we analyzed 4 important ones, namely - Split Criterion, Max tree depth, Max leaf nodes, Min samples split (Minimum number of samples a node should have, for it to split further). Two of the most popular splitting criterion for decision trees are:

- **Gini Impurity** [1] is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.
- **Entropy** [4] is an information theory metric that measures the impurity or uncertainty in a group of observations.

Details of other hyperparameters can be found on the Scikit-learn website.

In order to find an unbiased estimate of the performance of Decision Tree models with different hyperparameters, we have used cross-validation. Cross-validation [2] is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. This helps us tackle several biases such as sampling bias and data scooping bias. Specifically, we have used 10-fold cross-validation. In general, for k-fold cross-validation, the input data is split into k subsets of data (also known as folds). An ML model is trained on all but one (k-1) of the subsets, and then the model is evaluated on the subset that was not used for training. This process is repeated k times, with a different subset reserved for evaluation (and excluded from training) each time.

3 Background - Experiment Design

To reduce the search space for hyperparameter tuning, we have used Orthogonal Arrays (OA) [5]. There are two benefits for using OA based designs:

- It is a highly fractional orthogonal design that is based on a design matrix that allows us to consider a selected subset of combinations of multiple factors at multiple levels.
- Orthogonal arrays are balanced to ensure that all levels of all factors are considered equally. For this reason, the factors can be evaluated independently of each other despite the fractionality of the design.

In the OA design, only the main effects and two-factor interactions are considered, and higher-order interactions are assumed to be nonexistent.

In order to decide the values to explore for numerical hyperparameters, we decided to use MaxPro [6] designs to choose space-filling levels. Since there is no cost for changing factor levels in a computer experiment, we can focus on placing the design points in the experimental region in an intelligent manner. Therefore, by definition, a space-filling design should have points everywhere in the experimental region with as few gaps or holes as possible. We want to leverage this property of space-filling designs to select hyperparameters as this will allow us to "cover" the experimental region well.

There are many space-filling designs and the reader can read more about them here. Link - For this project, we used MaxPro design because of the following two reasons:

- It ensures good projections to all subspaces of the factors. Therefore, if some of the factors are not significant, the projections on other factors are also space filling.
- It has been shown empirically that MaxPro design does better than MmLHD for dimensions 3-9 and we have 3 factors in our design.

4 Data Used

For modeling purposes, MNIST dataset has been used. It is a very famous dataset for classification tasks. The MNIST database consists of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. For this project, we are just looking at cross-validation accuracy to compare Decision Tree Classifiers created for different hyperparameter values.

5 Analysis

We have created an OA based experiment design, and used the levels generated from MaxPro design for numerical variables (in addition to both levels for the categorical variable, splitting-criterion). This resulted in a hyperparameter search space with 16 runs, as compared to the total space which would have 128 runs (4 values for each of the 3 continuous hyperparameters and one binary hyperparameter). Also given than we are using 10-fold cross validation for stable accuracy metrics, we create 10 models at each run. Therefore we have reduced the number of models required from 1280 to 160. Generated design can be seen in Figure 2.

index	criterion	max_depth	max_leaf_nodes	min_samples_split
1	gini	3	60	0.00001
2	gini	9	1485	0.30000
3	gini	19	2000	0.10000
4	gini	24	592	0.20000
5	gini	9	592	0.10000
6	gini	3	2000	0.20000
7	gini	24	1485	0.00001
8	gini	19	60	0.30000
9	entropy	19	1485	0.20000
10	entropy	24	60	0.10000
11	entropy	3	592	0.30000
12	entropy	9	2000	0.00001
13	entropy	24	2000	0.30000
14	entropy	19	592	0.00001
15	entropy	9	60	0.20000
16	entropy	3	1485	0.10000

Figure 2: Generated Design

We used Python to run the decision tree models using hyperparameter combinations as per the given design. The cross-validation accuracy was measure for each run. This design along with accuracy values was used to calculate main-effects of each of the factors included in the model. The result from of ANOVA on the model is shown in Figure 3. As we can see that the main effect of the hyperparameter Minimum Sample Spilt is highly significant. We see in our analysis that the effect is negative, hence lower the value of Min-Samples-Split, better is the accuracy of our model.

```
Analysis of Variance Table

Response: mean_test_score
                              Df   Sum Sq  Mean Sq F value Pr(>F)
factor(param_criterion)        1 0.002506 0.002506  0.3129 0.6001
factor(param_max_depth)        3 0.057654 0.019218  2.3990 0.1840
factor(param_max_leaf_nodes)   3 0.020729 0.006910  0.8625 0.5181
factor(param_min_samples_split)  3 0.301239 0.100413 12.5347 0.0092 **
Residuals                      5 0.040054 0.008011
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 3: Results

6 Conclusion

From this analysis we were able to reduce the time required for execution from 49min 3s to 6min 36s, **an improvement of 7.43 times**. Given computer experiments in the world of Machine Learning can be very expensive (single run of a model can take upto days even). Being able to reduce the search space and then gain actionable insight from the reduced space is a valuable addition to the Hyperparameter Search methodologies currently in play. For example, on our dataset we found that only one hyperparameter is significant, we can now reduce the hyperparameter search space from 4 dimensions to 1, greatly speeding up the process. Advantage over the Random Hyperparameter Search is the information we gain from this approach (which can help us make other modeling decisions in our project too) and the certainty of a good result (which might not happen in Random searching).

References

[1] Gini Impurity @ https://bambielli.com/til/2017-10-29-gini impurity/.

[2] Cross Validation @ https://docs.aws.amazon.com/machine-learning/latest/dg/cross validation.html.

[3] Decision Tree Visualization @ https://www.mastersindatascience.org/learning/introduction-to-machine-learning-algorithms/decision tree/.

[4] Entropy @ https://www.section.io/engineering-education/entropy-information-gain-machine learning/.

[5] Orthogonal Arrays @ https://www.weibull.com/hotwire/issue131/hottopics131.html.

[6] V. Roshan Joseph. Space-filling designs for computer experiments: A review. *Quality Engineering*, 28(1), 1 2016.

[7] Decision Trees @ Scikit Learn. See more at https://scikit learn.org/stable/modules/tree.html.