COSC 5010-03 Practical Machine Learning Fall 2023 Hyperparameter Optimization Report

Michael Elgin

October 31, 2023

1 Introduction

Various machine learning algorithms can not only be trained on data but also tuned based on hyperparameters they offer. This can improve a model's ability to generalize to new data. This exercise compares the performance of various algorithms as their hyperparameters are adjusted. The wine quality dataset[1] is used.

2 Dataset Description

The wine quality dataset is standard tabular data. There are 2 datasets for each color of red and white. Both contain 11 features, all of which are continuous. The target is a discrete value which is the assigned quality of the wine. For the regression tasks, the white wine dataset is used to evaluate models. For classification tasks, both datasets are concatenated, with the target being changed to be the color of the wine, with 0 being assigned to red and 1 to white.

3 Experimental Setup

All computation is done with the Python programming language. Scikit-learn is used to construct models. Pandas is used to load and preprocess the data..

For regression, performance is the mean absolute percentage error, often graphed as its negative (then meaning more is better). This is defined by taking the difference between the predicted value of the model and the actual value, then dividing by the actual value. Then the mean of all of those is taken. More formally:

$$GE_{Regression\ model}(\hat{f}, \boldsymbol{X}_{test}, \boldsymbol{y}_{test}) = \frac{1}{|\boldsymbol{X}_{test}|} \sum_{i=1}^{|\boldsymbol{X}_{test}|} \frac{\hat{f}(\boldsymbol{X}_{test,i} - \boldsymbol{y}_{test,i}) \cdot 100}{\boldsymbol{X}_{test,i}}$$

For classification, the performance metric is standard accuracy:

[1] https://archive.ics.uci.edu/dataset/186/wine+quality

$$GE_{Classification\ model}(\hat{f}, \boldsymbol{X}_{test}, \boldsymbol{y}_{test}) = \frac{\sum_{i=1}^{|\boldsymbol{X}_{test}|} l_{0,1}(\hat{f}(\boldsymbol{X}_{test,i}), \boldsymbol{y}_{test,i})}{|\boldsymbol{X}_{test}|}$$

$$Acc_{Classification\ model} = (1 - GE_{Classification\ model}) * 100$$

For all models, grid-search is used for trying hyperparameter configurations. For hyperparameters that are continuous in nature, the grid is exponential in fashion, meaning exponents for 2^x are tried. This allows for a much wider exploration of the hyperparameter space given realistic time constraints, since adjusting hyperparameters in a linear fashion would space all configurations too closely together.

Grid search does not use nested resampling. For each hyperparameter config tested, it gathers the score at each cross validation split and as well as the average of those scores. When hyperparameter values are judged here in this report, it is based on the average of the averages of all times they were used. For example, if hypothetical hyperparameter $x = a$ (along with other hyperparameters and their values) is being evaluated, it is cross validated to produce one average, and then the true performance of value a is considered as the average of all the average scores whenever x was a even as other hyperparameter values were different.

The Bayesian optimization by itself does not use nested resampling. It uses a default of 3-fold inner cross-validation. As an additional layer to do nested resampling, 3-fold outer cross-validation wraps this process, and the average of those scores is reported in table 5.

The first section is regression models. Model 1 is a decision tree. The hyperparameters considered for this are max depth for the tree and minimum samples required for a split. All hyperparameters explored can only have positive numbers. The minimum amount of samples must be 2, hence the first exponent starts at 1.

$$\text{max depth} = 2^x, x \in [0, 7]$$

$$\text{min samples} = 2^x, x \in [1, 15]$$

The second model is a random forest, which uses the same hyperparamters as the tree but also adds in the third hyperparameter of the amount of trees in the forest.

$$\text{max depth} = 2^x, x \in [0, 4]$$

$$\text{min samples} = 2^x, x \in [1, 4]$$

$$\text{number of trees} = 2^x, x \in [0, 7]$$

The second section is classification models. Model 1 is a support vector classifier. Its first hyperparmeter is "C", which is inverse regularization strength. The second hyperparameter is the kernel type, which is either poly (polynomial) or rbf (radial basis function).

$$\text{C} = 2^x, x \in [0, 15]$$

$$\text{kernel type} \in \{\text{poly, rbf}\}$$

Model 2 is logistic regression, which uses the following hyperparameter settings.

$$\text{C} = 2^x, x \in [0, 8]$$

$$\text{penalty type} \in \{\text{l1, l2}\}$$

Model 3 is a decision tree classifier, which uses the same hyperparmeter settings as in regularization.

$$\text{max depth} = 2^x, x \in [0, 7]$$

$$\text{min samples} = 2^x, x \in [1, 15]$$

Model 4 is a K-nearest neighbor classifier, whose hyperparameters are the amount of neighbors to consider and the distance metric.

$$\text{number of neighbors} = 2^x, x \in [0, 7]$$

$$\text{distance metric} \in \{\text{l1, l2}\}$$

Model 5 is a random forest classifier, which uses the same hyperparameters as in regression.

$$\text{max depth} = 2^x, x \in [0, 4]$$

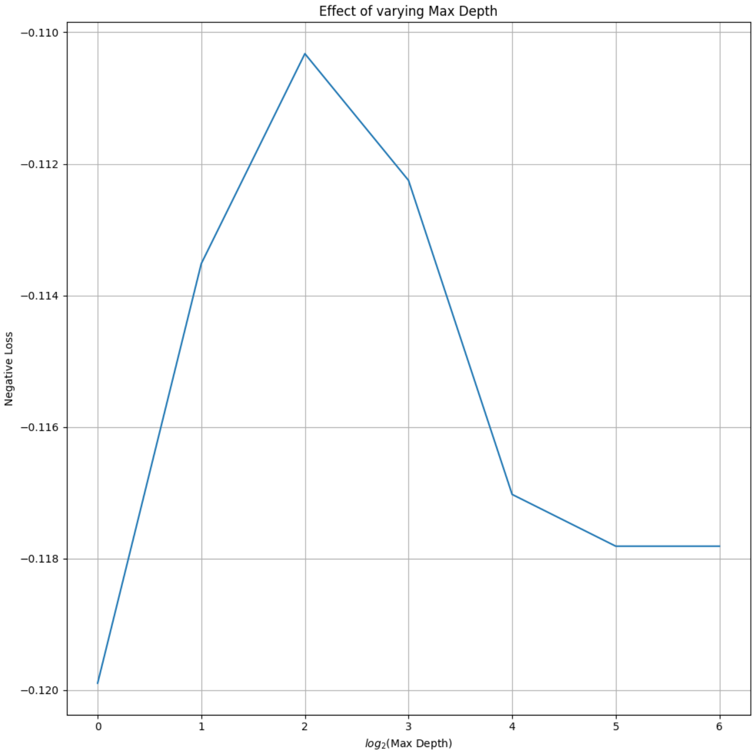
$$\text{min samples} = 2^x, x \in [1, 4]$$

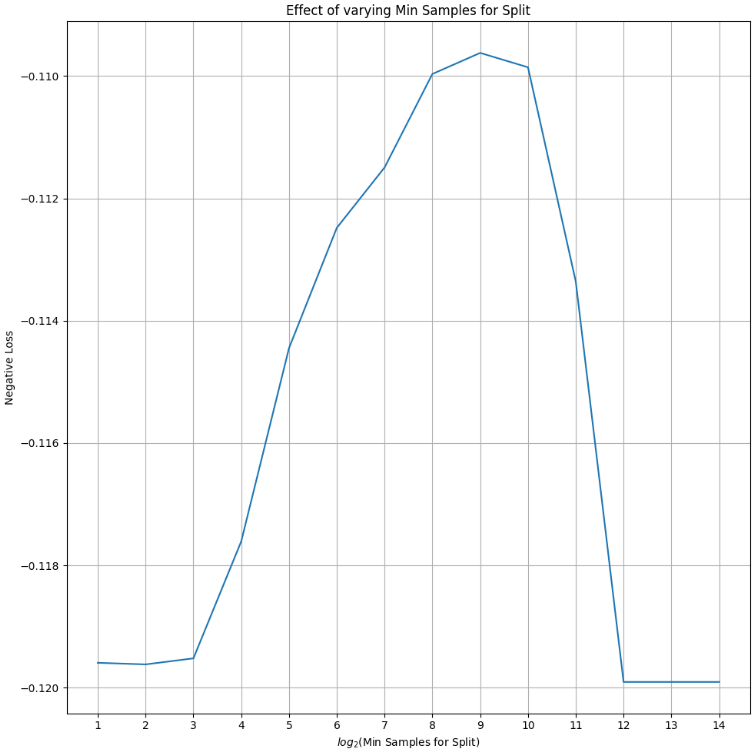
$$\text{number of trees} = 2^x, x \in [0, 7]$$

4 Results

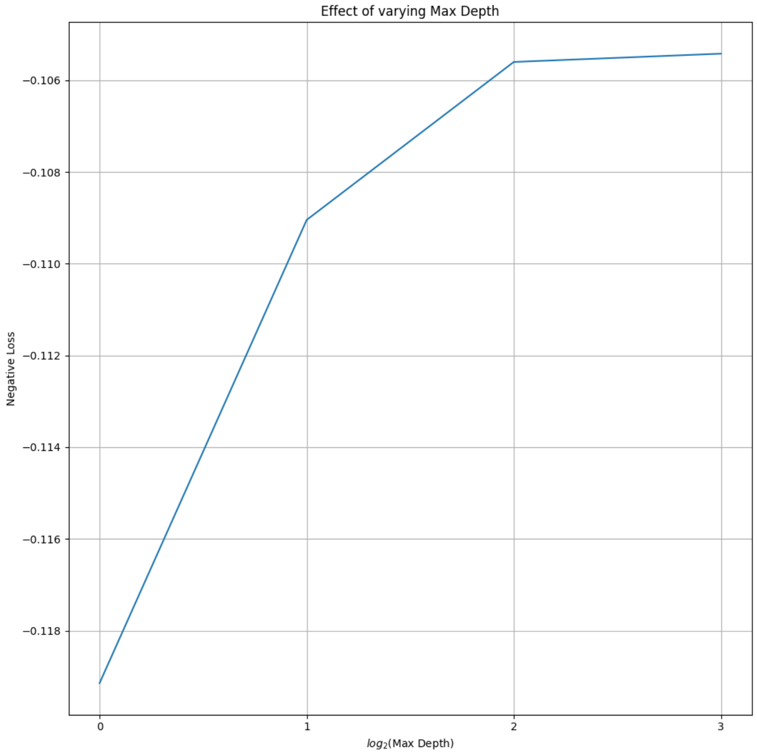
4.1 Plots of hyperparameter averages

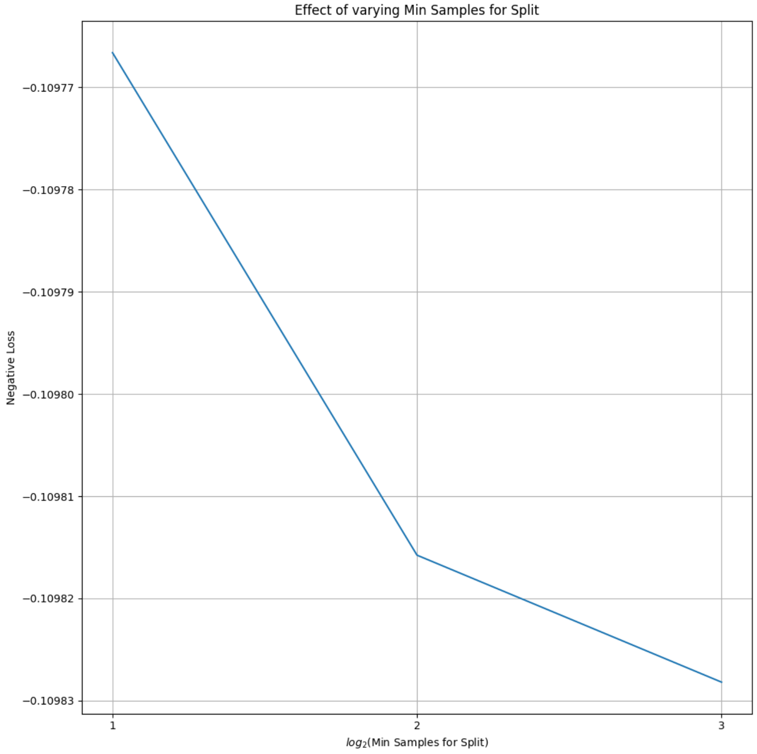
Plots for Decision Tree Regressor hyperparameters:

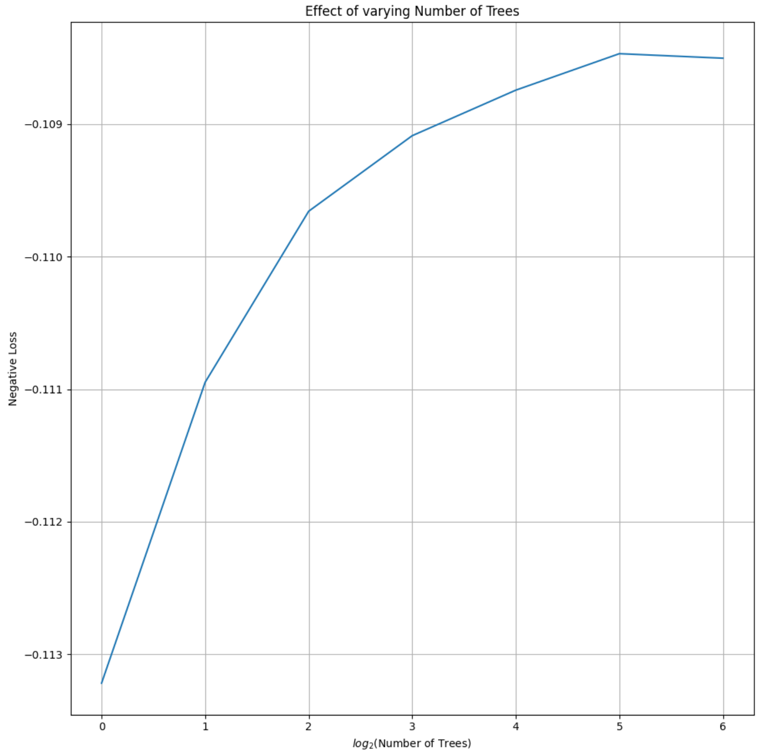




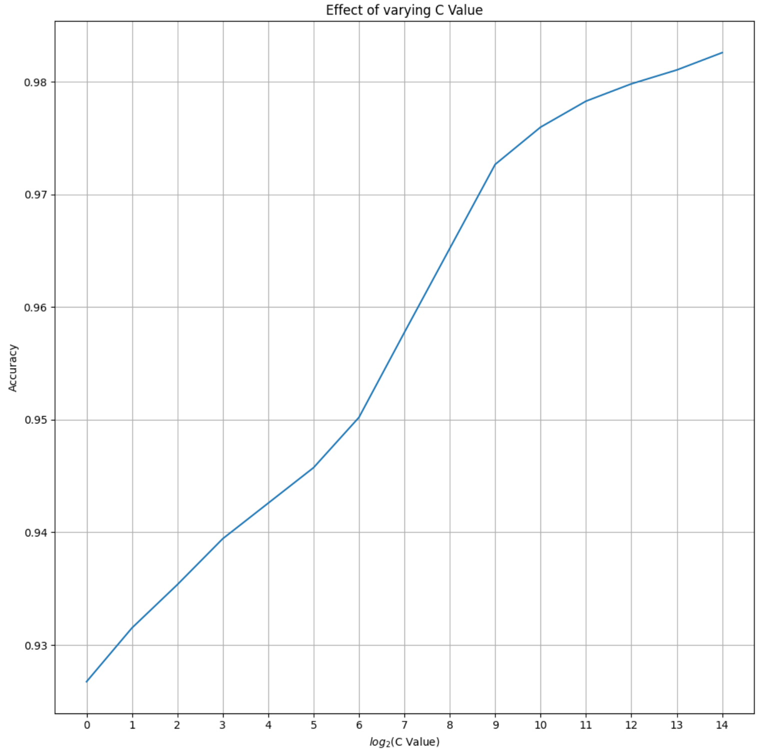
Plots for Random Forest Regressor hyperparameters:

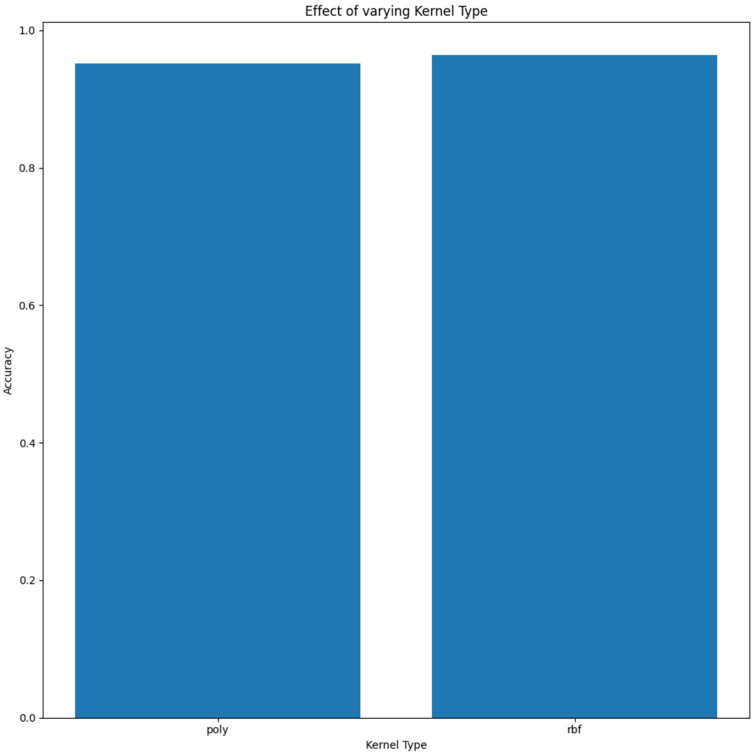

Effect of varying Max Depth
Negative Loss
−0.106
−0.108
−0.110
−0.112
−0.114
−0.116
−0.118
0
1
2
3
log_2(Max Depth)


Effect of varying Min Samples for Split
Negative Loss
−0.10977
−0.10978
−0.10979
−0.10980
−0.10981
−0.10982
−0.10983
1
2
3
log_2(Min Samples for Split)

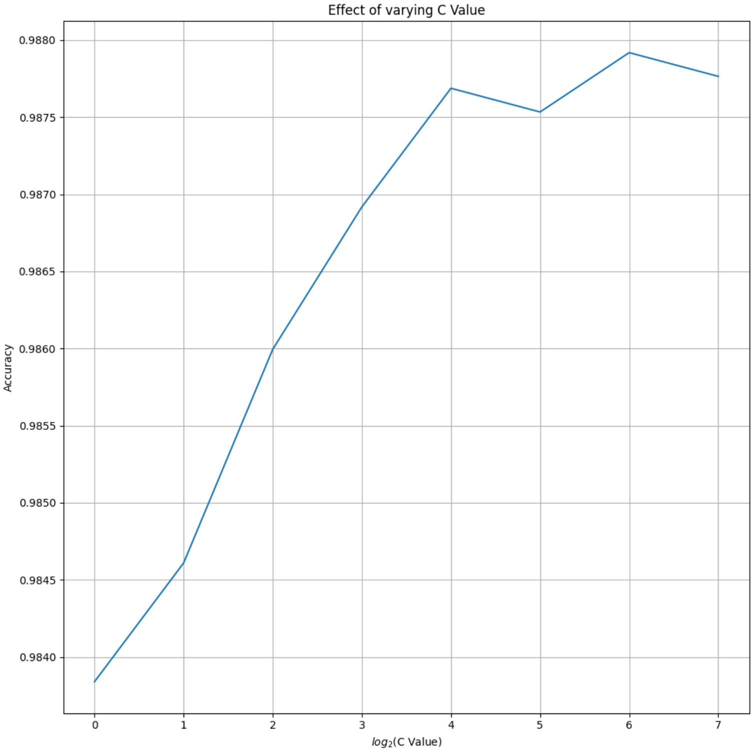


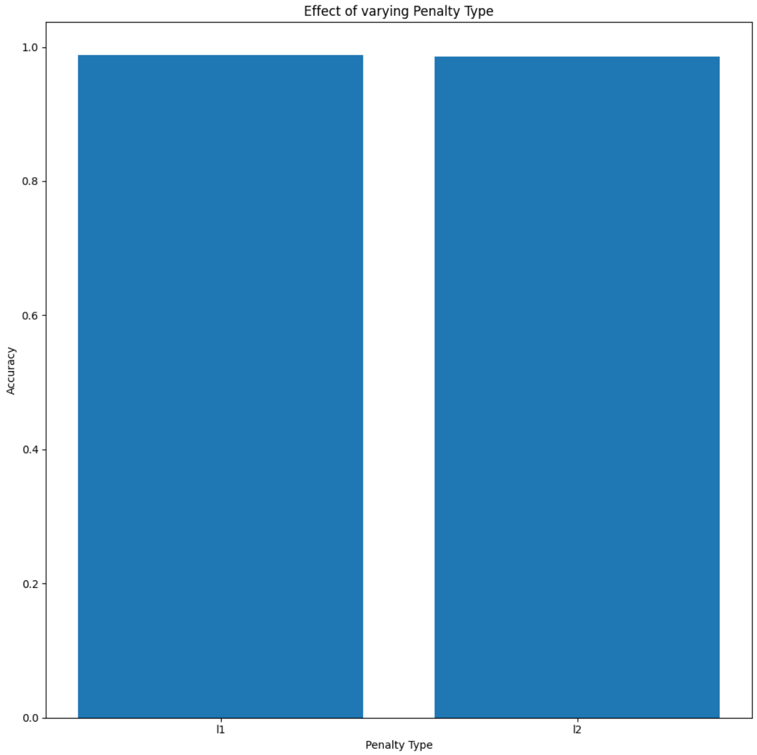
Plots for Support Vector Classifier hyperparameters:



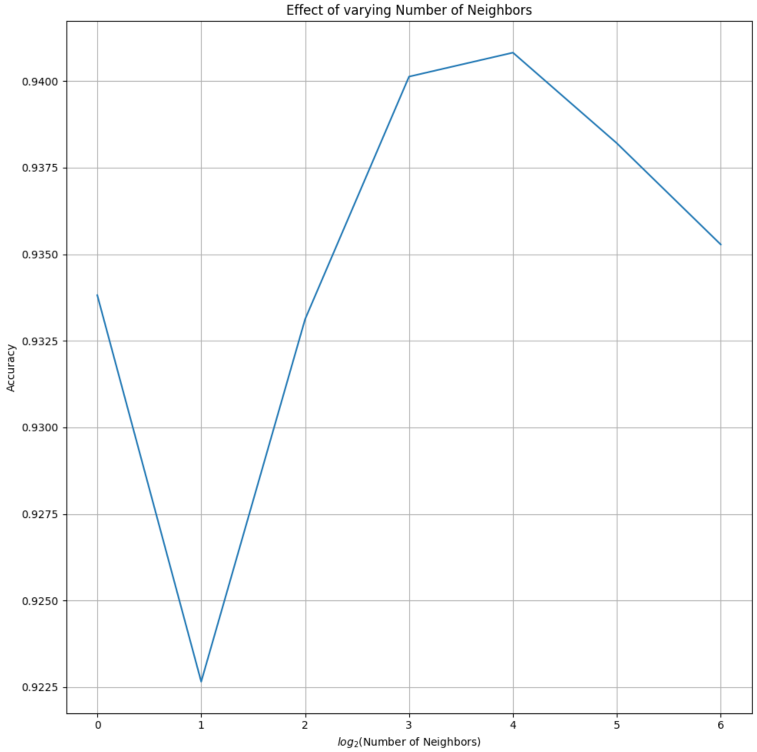


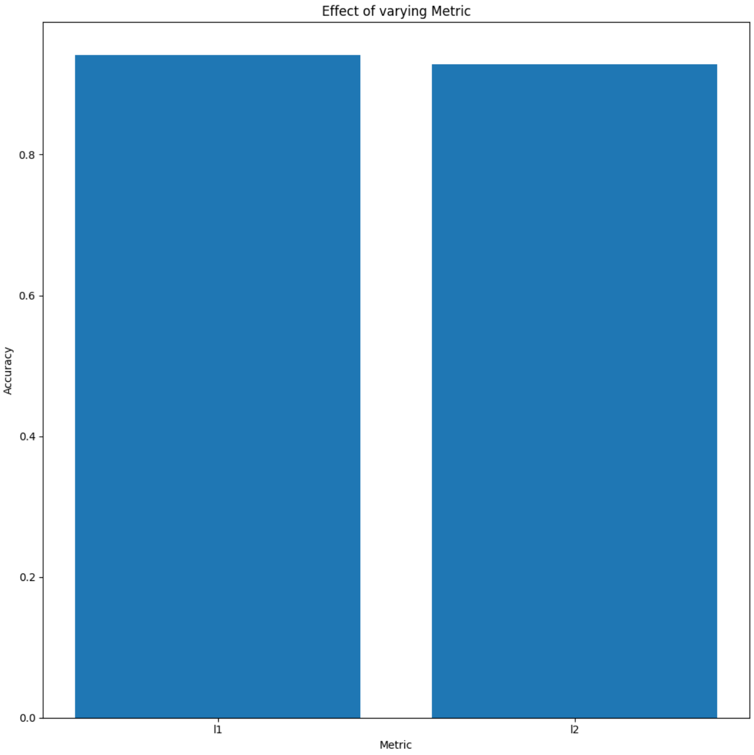
Plots for Logistic Regression hyperparameters:



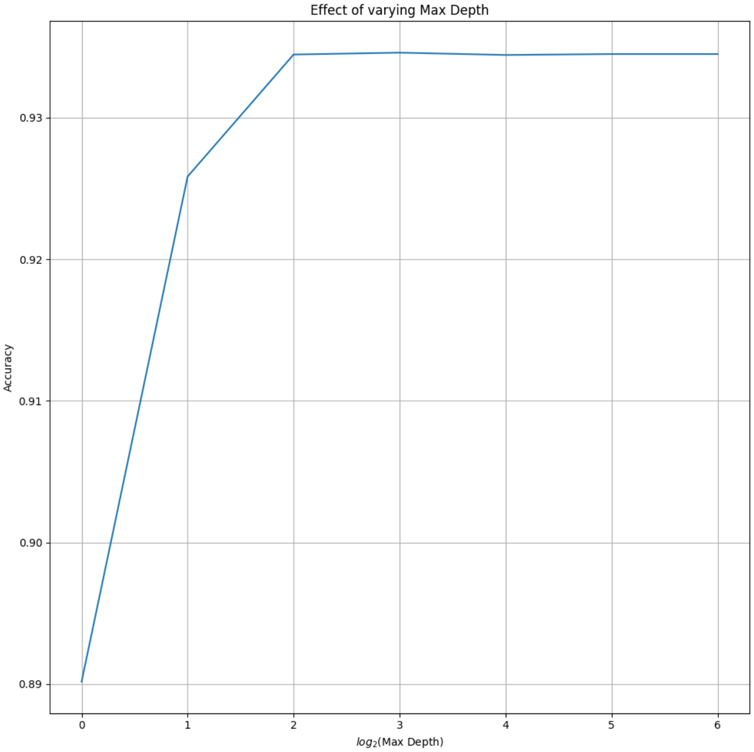


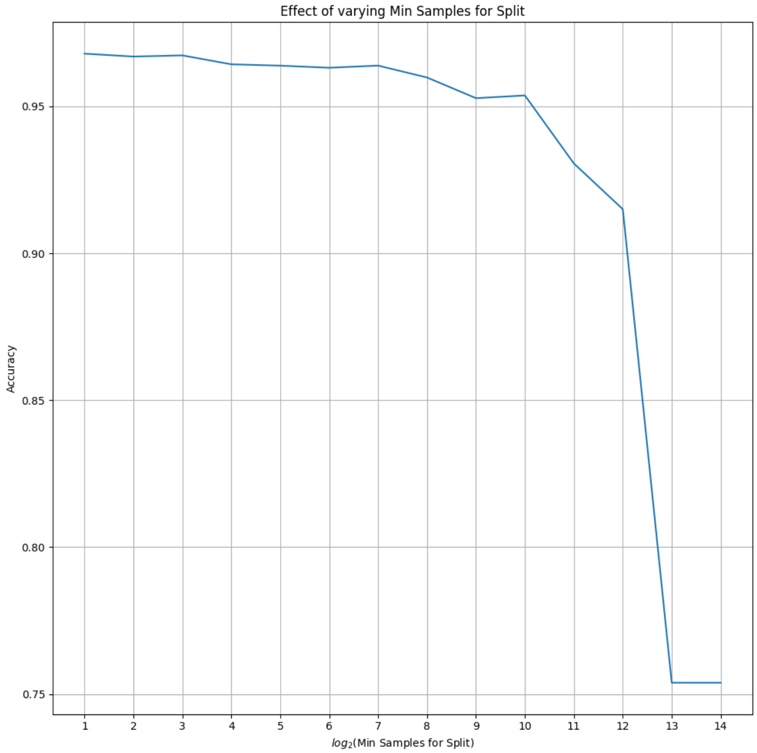
Plots for K-Nearest Neighbor hyperparameters:



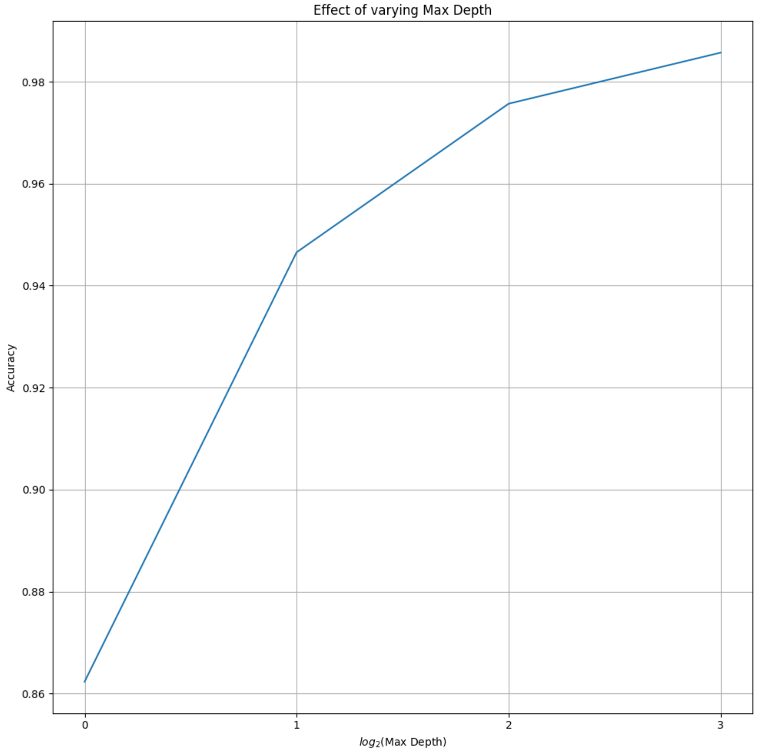


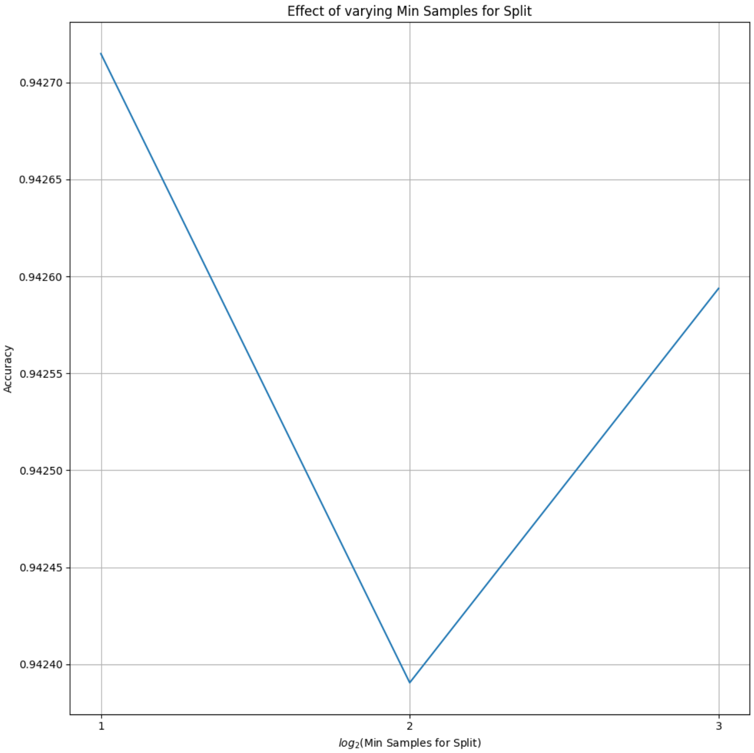
Plots for Decision Tree Classifier:

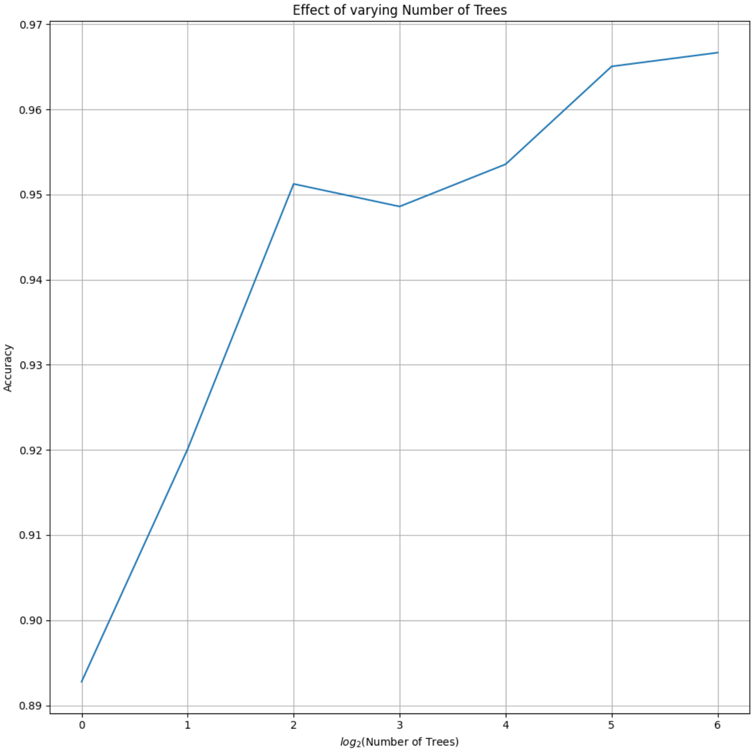




Plots for Random Forest Classifier:

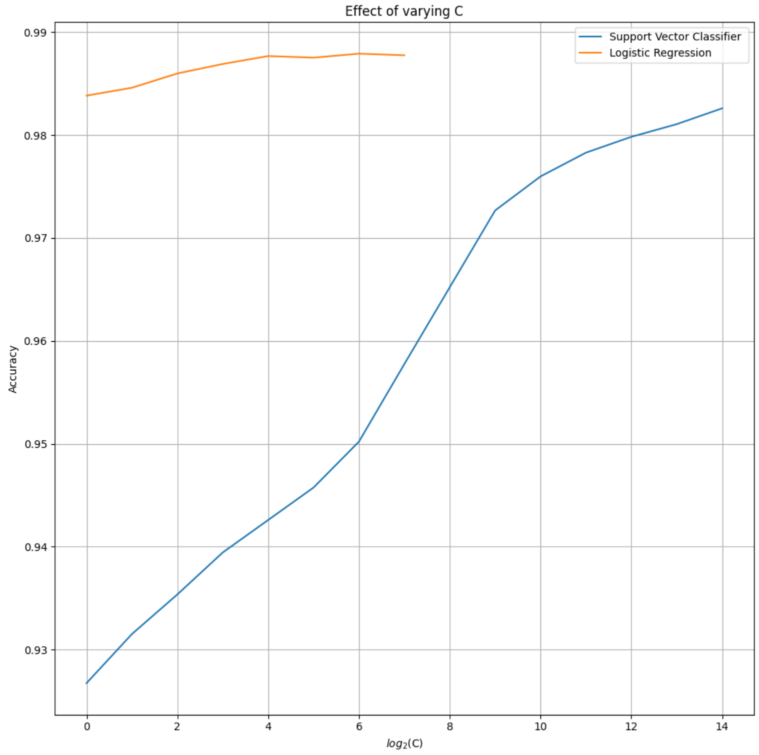


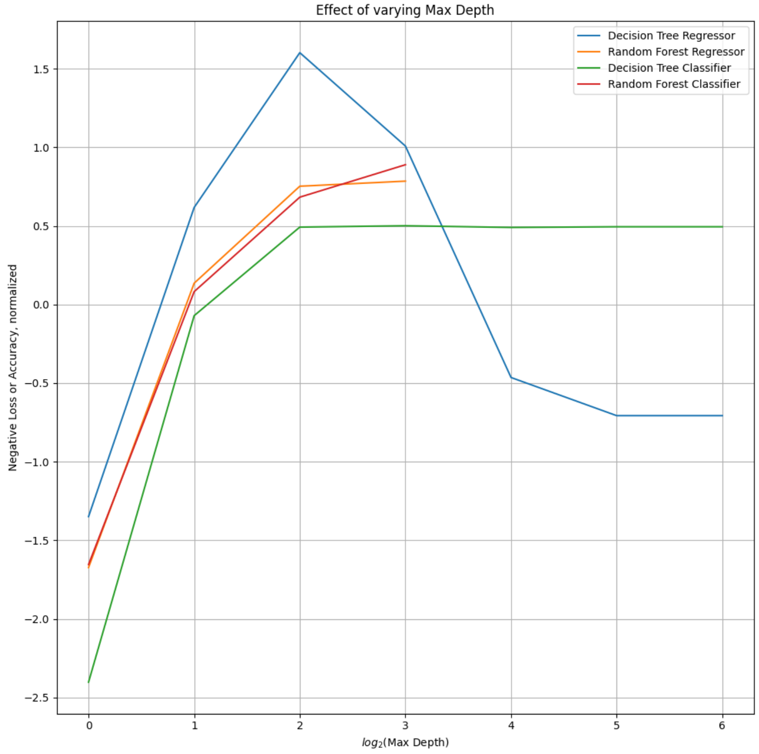


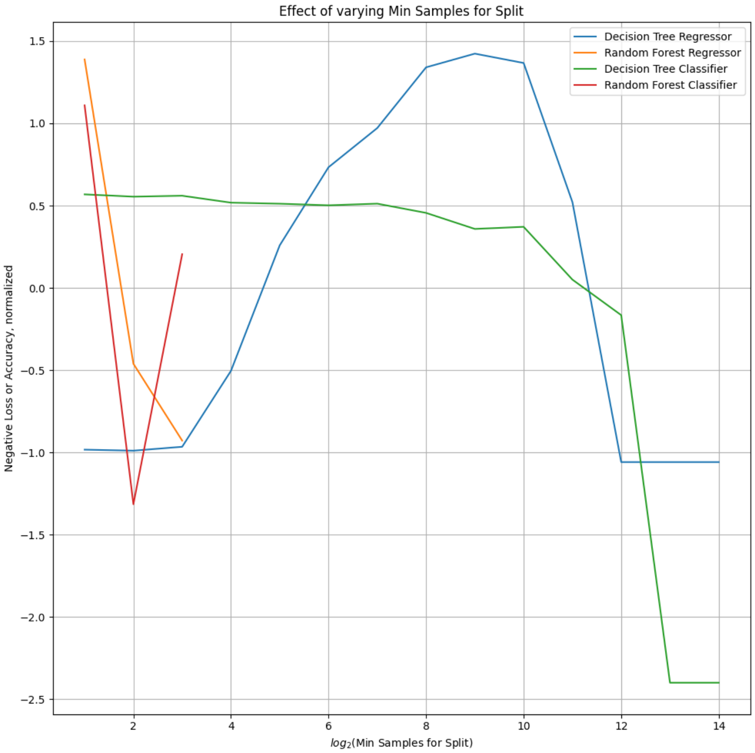


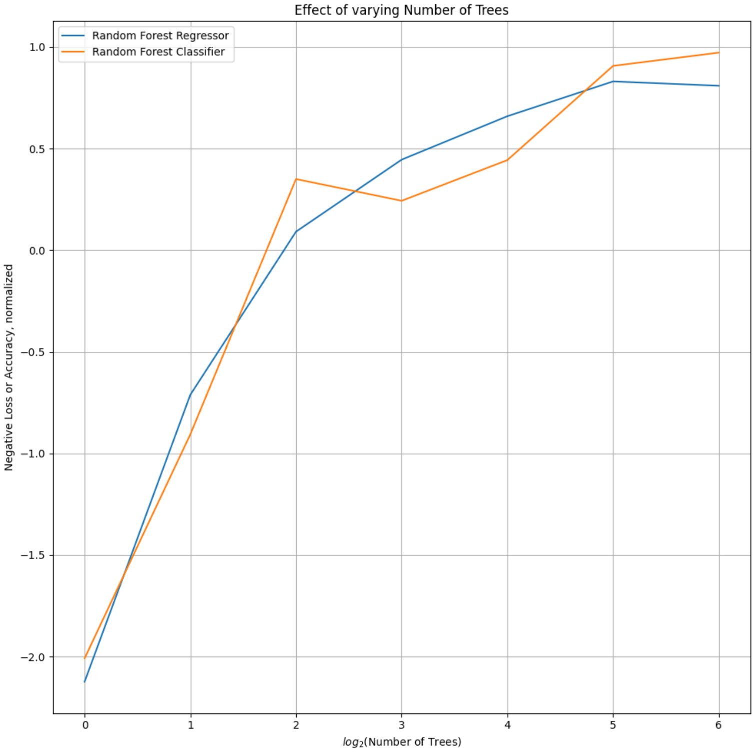
Combination plots follow[2].

[2] Note that the time cost of grid-search kept more values from being explored.


Effect of varying C
Support Vector Classifier
Logistic Regression
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0
2
4
6
8
10
12
14
log_2(C)


Effect of varying Max Depth
Decision Tree Regressor
Random Forest Regressor
Decision Tree Classifier
Random Forest Classifier
Negative Loss or Accuracy, normalized
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0
1
2
3
4
5
6
log_2(Max Depth)





For some hyperparameters, similar trajectories were observed between models. For others there was no reliable pattern across the space of values.

4.2 Tables of best hyperparameters

The following tables show the best configurations found from grid-search. These do not always match the best points on the plots of the averages because occasionally the combination of several hyperparameter settings that weren't optimal on average can still be the best when working together. The first value is from grid-search, the second is from Bayesian optimization. Note that results from runs of Bayesian optimization may vary if the random_state seed is changed from 0.

Table 1: Regression models' best hyperparameters

	Max Depth	Min Samples for Split	Number of Trees
Decision Tree	8, 44	512, 582	
Random Forest	8, 8	8, 8	64, 58

Table 2: Classification models' best hyperparameters

	C	Kernel Type	Penalty Type
Support Vector Classifier	16348, 15846	rbf, rbf	
Logistic Regression	64, 81		l2, l2

Table 3: Classification models' best hyperparameters

	Number of Neighbors	Distance Metric
K-Nearest Neighbor	8, 10	l1, l1

Table 4: Classification models' best hyperparameters

	Max Depth	Min Samples for Split	Number of Trees
Decision Tree	32, 58	2, 2	
Random Forest	8, 8	2, 8	32, 54

4.3 Table of performance measures for grid-search and Bayesian optimization

Table 5: Performance comparison for grid-search and Bayesian optimization

	Grid Search	Bayesian Optimization
Decision Tree Regressor	-0.1070	-0.1066
Random Forest Regressor	-0.1014	-0.1014
Support Vector Classifier	98.815%	98.784%
Logistic Regression	98.815%	98.846%
K-Nearest Neighbors	94.582%	94.890%
Decision Tree Classifier	98.107%	98.106%
Random Forest Classifier	99.338%	99.292%

Based on the table, it is not clear whether grid-search or Bayesian optimization is “better” than the other. There were some cases where each produced a final model that had slightly better performance than the other. In all cases, the differences were trivial.

5 Code

The associated code is in HPO.ipynb