Multi-Criteria Optimization

Finn Tomasula Martin
COSC-4557

1 Introduction

Most machine learning techniques aim to optimize models with respect to a single performance measure such as mean squared error in regression tasks or classification error in classification tasks. But, sometimes you may want to optimize models with respect to more than criteria. For example you may want to optimize a combination of performance and number of features used or performance and time taken to train. This technique is known as multi-criteria optimization. Multi-criteria optimization works by training models using any number of defined measures and then returning a thing called a Pareto front. A Pareto front will be a list of results from the optimization process where each member is not dominated by any other member. That means that for each member in the Pareto front, there is no other member found by the optimization process where all measures are better or equal. Each member of the Pareto front is considered equally good and it is up to the user to pick the member that most lines up with what they are trying to do. In this paper we will be examining how we can practically do multi-criteria optimization to get a Pareto front for several models.

2 Dataset Description

The dataset we will be examining for this exercise is called winequality-red.csv. It contains 1599 observations on various red wines and aims to predict “quality” of wine on a scale of 3 – 8. It uses 11 various features to make predictions. The dataset is fairly imbalanced across both the features and the target so we will scale the features and oversample the data by a factor of 6 to balance the classes.

3 Experimental Setup

We will be completing this analysis in R using the mlr3 framework. To start we load in our data and set our seed to 123 for reproducibility. Next we define a task for our wine dataset with quality as the target. Now we can move on to building our pipelines to be optimized. We define PipeOps for robust scaling and class balancing with a ratio of 6. Next we can define learners for each model we want to look at with various hyperparameters set to tune. Below is a table with the models we are looking at and the hyperparemter ranges for each:

Model	**Hyperparameters**
K-Nearest Neighbor	K = 1 - 20
	Distance = 1 - 20
Classification Tree	Max depth = 1 - 30
	Min bucket = 1- 50
Random Forest	Num trees = 100 – 1000
	Mtry = 1 - 8
	Min node size = 1 - 10

Next we will define the full pipeline for each model. Each pipeline will scale the data, balance the classes, stack a base k-nearest neighbor and random forest, and then tune the model of interest. Now we will define an optimization process for each pipeline. Each pipeline will be optimized using Bayesian optimization, run on our wine task, resampled using 10-fold cross validation, measured based on classification accuracy and time to train, and then run for 50 evaluations. Notice how we are using two measures rather than one. In this case we are looking to optimize a combination of performance and time taken. So, our Pareto fronts will contain results that found the best performance in the least amount of time. In most cases we would also need to include some form of nested resampling to avoid getting a biased performance estimate. But since multi-criteria optimization returns a set of configurations rather than a single best one, we do not need to do nested resampling. In addition because we are using 10-fold cross validation, each configuration is picked based off of average results between the folds not a single test train split. That means that our results will not be biased toward any one section of the data and takes them all into account during optimization.

---

4 Results

After running the optimization here is the Pareto front found for each model:

K-Nearest Neighbor

K	**Distance**	**Classification Error**	**Time to Train**
14	16.348	0.299	5.672
14	15.744	0.301	5.615
18	17.282	0.305	5.559
17	18.315	0.302	5.571
18	15.721	0.309	5.549

Classification tree

Max Depth	Min Bucket	Classification Error	Time to Train
11	6	0.291	5.597
9	14	0.305	5.505
11	5	0.298	5.554
9	6	0.298	5.562

Random Forest

Num Trees	Mtry	Min Node Size	Classification Error	Time to Train
111	1	1	0.300	5.634
100	1	10	0.296	5.770
108	1	10	0.299	5.711
102	1	10	0.298	5.736
125	1	10	0.297	5.737

From here it depends on what exactly your priorities are when picking a model and configuration. If you want your model to be as fast as possible, you can pick the configuration with the lowest time to train. If you want your model to be as accurate as possible, you can pick the configuration with the lowest classification error. Or, you can pick a model you feel balances the two. Overall, this technique is an effective way to train models with respect to more than one measure which can be a valuable tool in practice.

---

5 Sources

Schneider L, Becker M. (2024). Advanced Tuning Methods and Black Box Optimization. In Bischl B, Sonabend R, Kotthoff L, Lang M, (Eds.), *Applied Machine Learning Using mlr3 in R*. CRC Press. https://mlr3book.mlr-org.com/advanced_tuning_methods_and_black_box_optimization.html.

“Class for Multi Criteria Tuning - Tuninginstancemulticrit.” - *TuningInstanceMultiCrit • Mlr3tuning*, mlr3tuning.mlr-org.com/reference/TuningInstanceMultiCrit.html.

Casalicchio, Giuseppe. “Imbalanced Data Handling with MLR3.” *Mlr*, 30 Mar. 2020, mlr-org.com/gallery/basic/2020-03-30-imbalanced-data/.