Partitional Estimation Using Partial Moments

Viole, Fred
fviole@fordham.edu

April 18, 2020

Abstract

Function approximation is at the heart of machine learning. Given a dataset comprised of inputs and outputs, we assume that there is an unknown underlying function that is consistent in mapping inputs to outputs in the target domain and resulted in the dataset. We then use supervised learning algorithms to approximate this function. We highlight the identical objective of multivariate nonparametric regressions, for both continuous and discrete outputs (dependent variables), using both numerical and categorical inputs (regressors). Underlying the multivariate nonparametric technique is the use of partitional based conditional estimation. We examine 3 methods of partitioning and their resulting conditional estimates in evaluating unknown functional forms. We find the iterated means partitioning technique employed by the NNS R-package achieves superior mean-squared errors from the true forms in multivariate simulations.

1 Introduction

Nonlinear nonparametric statistics, henceforth "NNS", offers distinct capabilities to the modern econometrician's toolkit. Classical linear based approaches have long outstayed their welcome, inviting recent challenges from machine learning "ML" algorithms. However, the central objective of ML is that of prediction, while the econometrician is primarily concerned with explanation. It is this chasm that NNS seeks to fill, offering a general ML framework with interpretable results. This paper will introduce the use of partial moments underlying the NNS techniques, specifically, we will examine partition based estimation.

Why focus on partition based estimation? Quite simply, partition based estimation is at the heart of all ML algorithms [9]. The purpose of this paper is to demonstrate the ability of NNS to deliver reliable and robust partitional estimates, and compare NNS to more recent techniques in the literature [1,2].

Partition based estimation has been in the econometric toolkit for quite some time [8]. A histogram is indeed a partitional based estimation technique yet we do not rely upon it for inference and prediction. Smoothing of histograms to yield Nadaraya–Watson kernel densities was developed in the 1960s. More

accurate techniques have evolved based on the partition of the distribution of variables. Of particular note is the kernel based regression techniques built from these partitional building blocks [4, 7, 11]. These techniques have found their way into many applications as shown in [6], and given their pervasiveness, more theoretical work on robustness and asymptotics has been offered [1, 2, 3, 5, 10]. These results cover general data-driven partitioning methods, of which NNS belongs to.

The data-driven partitioning method classification is really the extent of association between NNS and other methods. ML techniques, such as random forests, neural networks, support vector machines,etc. derive their conditional expectations in very different manners. For example, random forests typically create a single partition in variables sequentially while neural networks handle variables simultaneously. We leave the comparison of NNS to existing ML methods in classification and continuous variable prediction to future work, though the extensions of the partitioning principles described here hold in those cases as well.

The remainder of this paper is as follows. The next section will offer the intuition behind the NNS partitions, and compare them to the alternative partitional methods. Using simulations of examples offered by other techniques, we will directly compare results. The following section will describe the simulations, and the following section after that will present the results. Finally, we will offer some comments on additional applications of these techniques. The contribution to the literature is a novel toolset robust to persistent concerns of the economist in part due to her limited toolkit. Linearity should be a pleasant surprise, not a prerequisite to modern econometric analysis. The partitional based estimation techniques shed this conundrum, and NNS warrants serious consideration in this space.

2 Intuition Behind NNS Partitions

NNS partitions were realized from a behavioral finance perspective on variance. Upside and downside variance have very different interpretations to the investor and implications to their individual utility functions. Parsing of the variance through partial moments is critical in capturing these nuances. When applied to joint distributions and multivariate analysis, the parsing extension holds to co-partial moments, and investor objective functions [12]. The generalization to partitional based estimation was realized from this procession.

Figure 1 illustrates the iterative partitioning in the left column and resulting regression estimates in the right column.

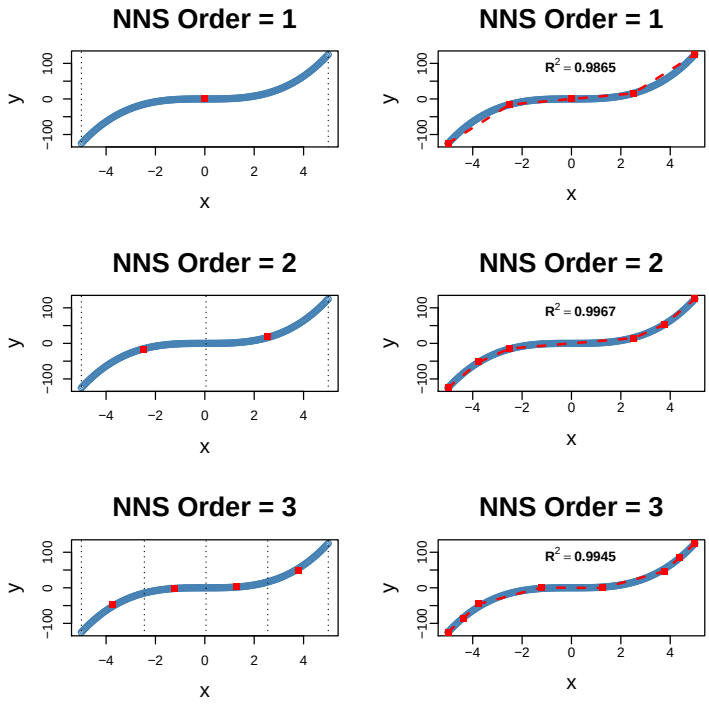

Figure 1. NNS partitioning and conditional estimates.

The NNS partitions use iterative means (red points in Figure 1), where conditional estimates are generated for each section left and right of the initial partition, then iterated on that current partition. This is equivalent setting a bandwidth that is used in kernel regression [9] or a number of quantiles used in other partitional based estimation methods [1]. A general characterization of the bias of partitioning-based estimators is provided in [2], and the NNS iterative mean partition falls under this generalization.[1] As in other kernel based regressions, NNS and its local linear segments will produce a consistent estimator of $f(x)$, since consistency is determined from bounded, symmetric, finite support conditions [9].

3 Number of Partitions

The number of partitions is paramount to making the conditional expectation operational, it is the direct counterpart of model selection for parametric approaches [9]. Not surprisingly, along with different methods of determining partitions, different objectives are offered for optimizing the number of partitions.

[1]Their definition of partition-based estimators, "First, the support X is partitioned into non-overlapping cells, which are then used to form a set of basis functions. Second, the final fit is determined by least squares regression using these bases."

The kernel based method uses least-squares cross validation to select an optimal bandwidth (h), where the ultimate objective function is to minimize the integrated squared area between $\int[\hat{f}(x) - f(x)]^2 dx$. This bandwidth is uniform throughout the support of X and its origins are from the finite difference difference method of derivatives when going from a CDF to estimate the PDF. Kernel based methods also heavily use interpolation (read made up data) for the density computations within each partition.

The least-squares partitions (J) presented in [1, 2] are always proportional guided by the number of observations (n) and order of the polynomial within the segment (p), such that $J = n^{\frac{1}{(2p+3)}}$. For linear segments where $p = 0$, the partitions reduce to $J = n^{1/3}$ and also try to reduce the integrated squared errors.

NNS partitions are iterated means, whereby partitioning stops when there are less than 10 observations in a subset. If there are but one or two observations in a subset, a linear regression could not be performed and it is suggested that between 10 and 20 observations are required to reasonably estimate effects [8]. NNS' objective is to minimize the sum of squared errors $\sum_{i=1}^{n}[\hat{f}(x_i) - f(x_i)]^2$.

These slight differences have large implications in conditional estimation as we shall see in our simulations to follow.

4 Simulations

The 3 partition based estimation methods considered will be Nonlinear Nonparametric Statistics (NNS v0.5.1), Nonparametric Estimation and Inference Procedures using Partitioning-Based Least Squares Regression (lspartition, v0.4), Nonparametric Kernel Smoothing Methods for Mixed Data Types (np, v0.60-10), and a random forest (randomForest v4.6-14) from their respective R packages (versions noted).

The simulations performed will evaluate the minimum mean squared error (MSE) of fitted values on the following multivariate regression functions with an added noise term consisting of a sample from uniform and normal distribution combined. The reason the error term was constructed in this manner was to not directly play into the underlying normality / symmetry assumption underlying within partition observation weighting for nonlinear kernels (especially since the real-world rarely, if ever, plays so nicely). A new seed is set for each of the 100 iterations allowing for reproducible random results.

Model 1: $y = sin(4x_1) + cos(x_2) - sin(x_1) * sin(3x_2)$

Model 2: $y = sin(\tau(x_1)\tau(x_2)\tau(x_3))$

where $\tau(x) = (x-0.5)+8(x-0.5)^2+6(x-0.5)^3-30(x-0.5)^4-30(x-0.5)^5$ and $n = 500$ for all models.

5 Results

The ratios of (NNS MSE / LSP MSE), (NNS MSE / np MSE) and the (NNS MSE / RF MSE) is presented below.

5.1 Model 1 Regression Results

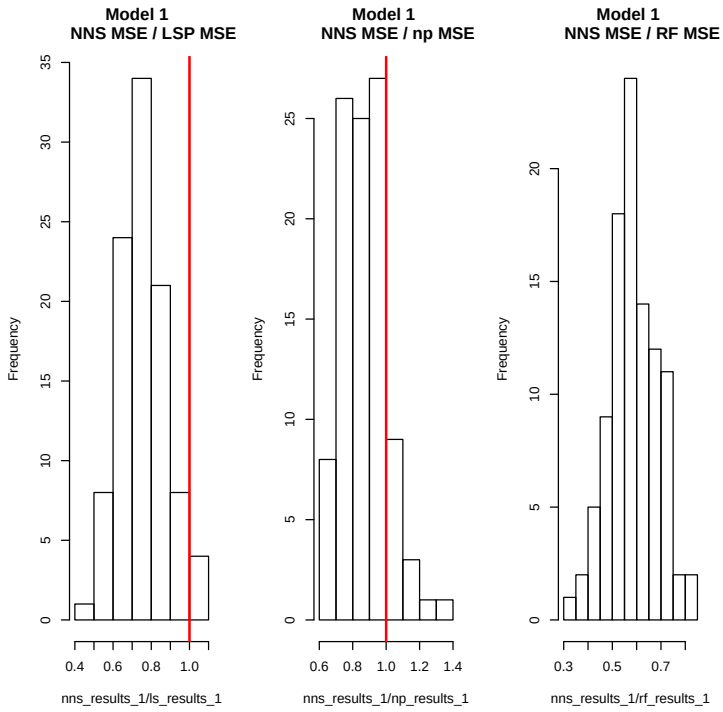

Figure 2. Results of simulations comparing MSE ratios for NNS, LSP, np and RF for regression model 1.

Statistically, looking at the significance of the differences of the MSE ratios (with a null hypothesis of no difference, a ratio of 1) yields the following results in Table 1. The NNS MSE is significantly lower than either the kernel regression, the least squares partition method, or the random forest.

	MSE Ratio	p value
NNS/LSP MSE	0.7578	0.0000
NNS/np MSE	0.8703	0.0000
NNS/RF MSE	0.5876	0.0000

Table 1: `Model 1 MSE ratios and p-values.`

5.2 Model 2 Regression Results

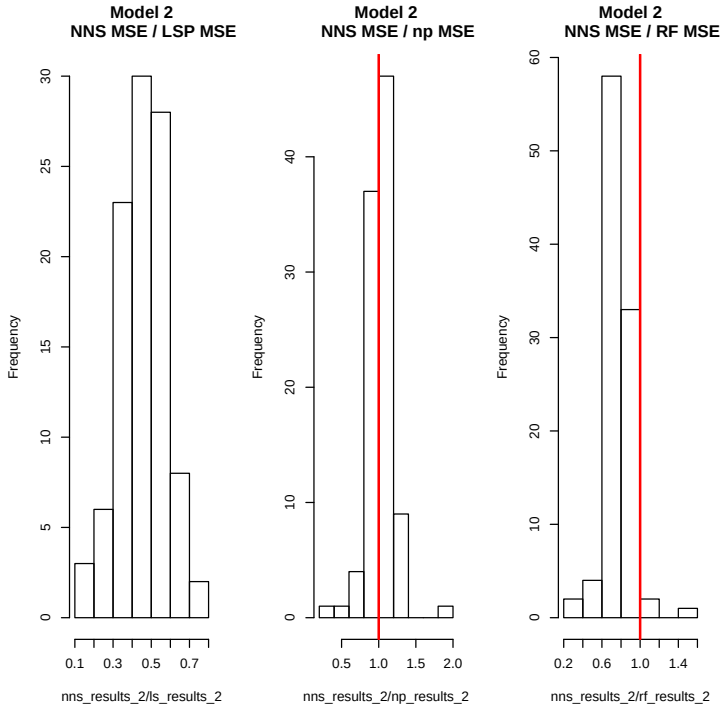

Figure 3. Results of simulations comparing MSE ratios for NNS, LSP, np and RF for regression model 2.

Statistically, looking at the significance of the differences of the MSE ratios (with a null hypothesis of no difference, a ratio of 1) yields the following results in Table 2. The NNS MSE is significantly lower than the least squares partition and random forest, but not statistically different from the kernel method.

	MSE Ratio	p value
NNS/LSP MSE	0.4576	0.0000
NNS/np MSE	1.0153	0.3819
NNS/RF MSE	0.7647	0.0000

Table 2: `Model 2 MSE ratios and p-values.`

6 Discussion and Comments

Partition based estimation is an exceptionally powerful method underlying nonparametric statistics and ML. The difference in results presented in the methods considered here lies in the choice of the number of partitions and treatment of endpoints in the support of observed X. None of the methods were grossly inadequate in their MSE estimates of the known true functional forms which supports the general characterization of the bias of partitioning-based estimators.

We can test an infinite number of functional forms in the presence of noise over an infinite number of iterations to be even more certain of the robustness of these estimation techniques. However, current ML methods are not as accurate as the partitional estimation techniques, yet they flourish in practice with increasing acceptance in central banks. Why is this? Because ML techniques give fast, scalable, seemingly acceptable answers to a variety of different questions where functional forms are never and can never be known, putting the onus squarely on predictive ability.

Most, if not all, current ML techniques do not extrapolate well beyond the support of observed X, which, along with partial derivative information, is where nonparametric regressions can fill the void. In order for these partitional based estimation techniques to compete with their ML offspring, additional programming insights are sorely needed. The more observations, the better the partitional estimates, yet they become computationally infeasible for large sample sizes. By highlighting the proficiency of these theoretically superior estimation techniques, attention will be brought to their cause and further computational advancements on the ever expanding and changing datasets policy makers now have at their disposal.

References

[1] Cattaneo, M. D., R. K. Crump, M. H. Farrell, and Y. Feng (2019): "Binscatter Regressions," in preparation for the Stata Journal.

[2] Cattaneo, M. D., M. H. Farrell, and Y. Feng (2018): "Large Sample Properties of Partitioning-Based Estimators," arXiv:1804.04916.

[3] Cattaneo, M. D., and M. H. Farrell (2011): "Efficient Estimation of the Dose Response Function under Ignorability using Subclassiffication on the Covariates," in Advances in Econometrics: Missing Data Methods, ed. by D. Drukker, vol. 27A, pp. 93-127. Emerald Group Publishing Limited.

[4] Cochran, W. G. (1968): "The Effectiveness of Adjustment by Subclassiffication in Removing Bias in Observational Studies," Biometrics, 24(2), 295-313.

[5] Calonico, S., M. D. Cattaneo, and R. Titiunik (2014): "Robust Nonparametric Confidence Intervals for Regression-Discontinuity Designs," Econometrica, 82(6), 2295-2326.

[6] Fama, E. F. (1976): Foundations of Finance: Portfolio Decisions and Securities Prices. Basic Books, New York, NY.

[7] Friedman, J. H. (1977): "A Recursive Partitioning Decision Rule for Nonparametric Classification," IEEE Transactions on Computers, C-26(4), 404-408.

[8] Hastie, T., R. Tibshirani, and J. Friedman (2009): The Elements of Statistical Learning, Springer Series in Statistics. Springer-Verlag, New York.

[9] Kleven, H. J. (2016): "Bunching," Annual Review of Economics, 8, 435-464.

[10] Vinod, H. and Viole, F. "Nonparametric Regression Using Clusters" *Comput Econ* (2018) 52:1317–1334 https://doi.org/10.1007/s10614-017-9713-5

[11] Viole, F. and Nawrocki, D. "The Utility of Wealth in an Upper and Lower Partial Moment Fabric" *The Journal of Investing*, Summer 2011, Vol. 20, No. 2: pp. 58-85.

Appendix

The R code used in the simulations:

```
> library(NNS)
> library(lspartition)
> library(np)
> library(randomForest)
> results <- list()
> cores <- detectCores()
> cl <- makeCluster(cores[1]-1)
> registerDoParallel(cl)
> results <- list()
> results <- foreach(i = 1:100, .packages=c("NNS", "data.table", "lspartition", "np",
                                            "randomForest"))%dopar%{

 # DGP
 set.seed(12345+i)

 tau.fx <- function(x){
   (x - 0.5) + 8*(x - 0.5)^2 + 6*(x - 0.5)^3 - 30*(x - 0.5)^4 - 30*(x - 0.5)^5
 }

 n <- 500
 x <- data.frame(runif(n), runif(n), runif(n))
 y_1 <- sin(4*x[,1]) + cos(x[,2]) - sin(x[,1])*sin(3*x[,2]) + rnorm(n)+runif(n)
 y_2 <- sin((tau.fx(x[,1])*tau.fx(x[,2])*tau.fx(x[,3]))) + rnorm(n)+runif(n)

 # lspartition model
 est_1 <- lsprobust(y_1, x[,1:2], eval = x[,1:2])
 est_2 <- lsprobust(y_2, x, eval = x)


 # NNS cross-validated model
 nns_est_1 <- NNS.stack(x[,1:2], y_1, ncores = 1)
 nns_est_2 <- NNS.stack(x, y_2, ncores = 1)

 # np model
 np_bw_1 <- npregbw(y_1 ~ x[,1] + x[,2])
 np_fit_1 <- npreg(np_bw_1, txdat = x[,1:2], tydat = y_1, regtype = "ll")$mean
 np_bw_2 <- npregbw(y_2 ~ x[,1] + x[,2] + x[,3])
 np_fit_2 <- npreg(np_bw_2, txdat = x, tydat = y_2, regtype = "ll")$mean


 # RF Model
 rf_est_1 <- randomForest(x = x[,1:2], y = y_1)$predicted
```

```
rf_est_2 <- randomForest(x = x, y = y_2)$predicted

# Results
fits <- cbind.data.frame(
  true_1 = sin(4*x[,1]) + cos(x[,2]) - sin(x[,1])*sin(3*x[,2]),
  true_2 = sin((tau.fx(x[,1])*tau.fx(x[,2])*tau.fx(x[,3]))),
                        nns_1 = nns_est_1$stack,
                        nns_2 = nns_est_2$stack,
                        lspartition_1 = est_1$Estimate[,4],
                        lspartition_2 = est_2$Estimate[,5],
                        np_1 = np_fit_1,
                        np_2 = np_fit_2,
                        rf_1 = rf_est_1,
                        rf_2 = rf_est_2)


# NNS MSE
results$nns_mse_1 <- mean((fits$nns_1 - fits$true_1)^2)
results$nns_mse_2 <- mean((fits$nns_2 - fits$true_2)^2)

# lspartition MSE
results$ls_mse_1 <- mean((fits$lspartition_1 - fits$true_1)^2)
results$ls_mse_2 <- mean((fits$lspartition_2 - fits$true_2)^2)

# np MSE
results$np_mse_1 <- mean((fits$np_1 - fits$true_1)^2)
results$np_mse_2 <- mean((fits$np_2 - fits$true_2)^2)

# RF MSE
results$rf_mse_1 <- mean((fits$rf_1 - fits$true_1)^2)
results$rf_mse_2 <- mean((fits$rf_2 - fits$true_2)^2)

return(results)
}
> stopCluster(cl)
> registerDoSEQ()
> nns_results_1 <- unlist(lapply(results, `[[`, 1))
> nns_results_2 <- unlist(lapply(results, `[[`, 2))
> ls_results_1 <- unlist(lapply(results, `[[`, 3))
> ls_results_2 <- unlist(lapply(results, `[[`, 4))
> np_results_1 <- unlist(lapply(results, `[[`, 5))
> np_results_2 <- unlist(lapply(results, `[[`, 6))
> rf_results_1 <- unlist(lapply(results, `[[`, 7))
> rf_results_2 <- unlist(lapply(results, `[[`, 8))
```