

OPTIMAL SAMPLING FOR STATISTICAL MODELING AND VALIDATION

A Dissertation
Presented to
The Academic Faculty

By

Akhil Vakayil

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology

May 2023

© Akhil Vakayil 2023

OPTIMAL SAMPLING FOR STATISTICAL MODELING AND VALIDATION

Thesis committee:

Dr. Roshan Joseph, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Ashwin Pananjady
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Santanu Dey
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Agus Sudjianto
Corporate Model Risk
Wells Fargo

Dr. Mohit Singh
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: May 5, 2023

To my parents and sister for their unwithering support

ACKNOWLEDGMENTS

I am sincerely grateful to my advisor, Dr. Roshan Joseph, for believing in my strengths and offering me a wonderful learning experience throughout my Ph.D. journey. I truly value my growth as an individual under his mentorship, and will forever cherish my research experience as a Ph.D. student.

I would like to thank all the members of my thesis committee for their help in preparation of this work. I also extend my wholehearted appreciation to my family and friends who have helped me achieve this important milestone in my life.

I gratefully acknowledge U.S. National Science Foundation grants CMMI-1921646 and DMREF-1921873 for supporting the research presented in this thesis.

TABLE OF CONTENTS

Acknowledgments	iv
List of Tables	viii
List of Figures	ix
Chapter 1: An Optimal Method for Data Splitting	1
1.1 Introduction	1
1.2 Methodology	2
1.2.1 Mathematical Formulation	2
1.2.2 Review of Data Splitting Methods	4
1.2.3 Support Points	7
1.2.4 SPlit	8
1.2.5 Visualization	10
1.2.6 Simulations	12
1.3 Categorical Variables	14
1.4 Examples	16
1.4.1 Regression	17
1.4.2 Classification	19
1.5 Conclusions	22

Chapter 2: Data Twinning	24
2.1 Introduction	24
2.2 Review of SPlit	26
2.3 Twinning	30
2.3.1 Algorithm	31
2.3.2 Computational Complexity	34
2.4 Experiments	36
2.5 Multiplets	39
2.6 Applications	41
2.6.1 Data Splitting	41
2.6.2 Data Compression	42
2.6.3 Cross Validation	44
2.7 Conclusions	46
Chapter 3: A Unified Global-Local Gaussian Process Approximation	48
3.1 Introduction	48
3.2 Gaussian Process Review	51
3.3 Global-Local Gaussian Process	52
3.3.1 Global and Local Points	53
3.3.2 Predictive equations	54
3.3.3 Correlation Functions	56
3.3.4 Computational complexity	58
3.4 1d Illustration	59

3.5	Experiments	61
3.5.1	Evaluation criteria	62
3.5.2	Emulation	63
3.5.3	Real world data	65
3.6	Conclusions	69
	Appendices	70
	Appendix A: Proofs	71
	References	73

LIST OF TABLES

1.1	Description of datasets considered for regression.	18
1.2	Description of datasets considered for classification.	21
2.1	Time taken to reduce the NYC taxi trip training set, and then fitting a random forest regression, on a laptop with 6-core Intel 2.6 GHz processor. Fields marked with an * are estimates.	44

LIST OF FIGURES

1.1	Comparison of CADEX and DUPLEX testing sets with SPlit testing set. 100 testing points (red circles) are chosen from 1,000 data points (black crosses). In the lower panels, the marginal densities of the testing sets are plotted over the histogram of the full data.	5
1.2	The circles are the testing set obtained using random (left) and SPlit (right) subsampling.	11
1.3	The squares are the validation set obtained using random (left) and SPlit (right) subsampling from the training set. The testing set is shown as circles.	12
1.4	The error in the estimation of generalization error with splitting ratios of 10% (left) and 50% (right) for different data splitting methods over 100 iterations.	14
1.5	The circles are the testing set obtained using naive method (left), stratified proportional method (center), and the proposed coding-based method (right). The three categorical levels are shown as red triangles, green pluses, and blue crosses.	15
1.6	Distribution of root mean squared error (RMSE) over 500 random and SPlit subsampling splits for the concrete compressive strength dataset.	18
1.7	Distribution of root mean squared error (RMSE) over 500 random and SPlit subsampling splits for the datasets described in Table 1.1.	19
1.8	Visualizing a SPlit subsampling testing set (circles) for the Iris dataset.	20
1.9	Distribution of residual deviance (D) over 500 random, stratified proportional random, and SPlit subsampling splits for the Iris dataset.	21
1.10	Distribution of residual deviance (D) over 500 random, stratified proportional random, and SPlit subsampling splits for the datasets described in Table 1.2.	22

2.1	Computation times for making an 80-20 split of d -dimensional multivariate normal datasets with N rows, using 500 iterations of the original algorithm for <code>SPlit</code> , on a 36-core processor.	26
2.2	The convex hull of subsets identified by <code>Twinning</code> at the end of iterations 1, 2, 5, and 10 for the sample dataset described in Section 2.3. Points in \mathcal{D}^1 are shown as encircled points, and they are numbered in the order they were selected.	34
2.3	A random split of the dataset described in Section 2.3 is shown on the left, while a <code>SPlit</code> obtained using DC-NN is shown in the middle, and with <code>Twinning</code> on the right. The encircled points constitute \mathcal{D}^1 in each of the three plots.	35
2.4	Comparison between DC-NN and <code>Twinning</code> for varying N with fixed d and γ . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and <code>Twinning</code> , respectively. The green crosses denote the energy ratio of <code>Twinning</code> to DC-NN.	37
2.5	Comparison between DC-NN and <code>Twinning</code> for varying d with fixed N and γ . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and <code>Twinning</code> , respectively. The green crosses denote the energy ratio of <code>Twinning</code> to DC-NN.	38
2.6	Comparison between DC-NN and <code>Twinning</code> for varying γ with fixed N and d . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and <code>Twinning</code> , respectively. The green crosses denote the energy ratio of <code>Twinning</code> to DC-NN.	38
2.7	Pictorial representation of the three proposed strategies to generate quadruplets of the same cardinality with <code>Twinning</code>	40
2.8	Distribution of $\overline{\text{ED}}_{n,N}^{1,2(*)}$ over 250 multi-splits of the multivariate normal dataset by random splitting, and the three proposed strategies for generating multi-plets with <code>Twinning</code>	41
2.9	Distribution of testing RMSE over 50 repetitions of compressing the NYC taxi trip training set with random subsampling, SDC-NN ($\kappa = 10$), and <code>Twinning</code> , followed by fitting a random forest regression model.	45
2.10	Distribution of λ_{min} (left) and MSCV (right) at λ_{min} , as estimated by LASSO with 4-fold and 8-fold cross validation, over 500 distinct random folds and multi-plets of the airfoil self-noise dataset.	46

3.1	Illustration of the training points (global and local) identified for a given training data and testing location with TwinGP.	55
3.2	Prediction and 2σ confidence interval (shaded region) with full GP, laGP, SGPR, and TwinGP to model the $1d$ function in (3.25).	60
3.3	Distribution of RMSE and NLPD over 50 iterations of modeling the piston simulation function with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	64
3.4	Distribution of RMSE and NLPD over 50 iterations of modeling the borehole function with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	65
3.5	Distribution of RMSE and NLPD over 50 iterations of modeling the Dette-Pepelyshev function with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	65
3.6	Distribution of RMSE and NLPD over 50 iterations of modeling the ozone column dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	66
3.7	Distribution of RMSE and NLPD over 50 iterations of modeling the protein structure dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	67
3.8	Distribution of RMSE and NLPD over 50 iterations of modeling the sarco robotics dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	68
3.9	Distribution of RMSE and NLPD over 50 iterations of modeling the flight delays dataset with laGP and TwinGP. The corresponding average computation time per iteration is given in the right most plot.	69

SUMMARY

In statistics and machine learning, often we need to sample from or partition data, e.g., for generating training-testing splits, subsampling for tractable statistical analysis, etc. This thesis presents an optimal sampling/partitioning methodology and its applications. Chapter 1 provides the motivation behind the proposed methodology from a validation perspective, Chapter 2 gives an efficient algorithm that makes the optimal sampling applicable to large datasets, and finally, Chapter 3 presents a novel Gaussian process approximation exploiting the proposed sampling methodology.

In Chapter 1, we propose an optimal method, referred to as `SPLit`, for splitting a dataset into training and testing sets. `SPLit` is based on support points (SP), which was initially developed for finding the optimal representative points of a continuous distribution. We adapt SP for subsampling from a dataset using a sequential nearest neighbor algorithm. We also extend SP to deal with categorical variables so that `SPLit` can be applied to both regression and classification problems. The implementation of `SPLit` on real datasets shows substantial improvement in the worst-case testing performance for several modeling frameworks, compared to the commonly used random splitting procedure.

In Chapter 2, we develop a method named `Twinning` for partitioning a dataset into statistically similar twin sets. `Twinning` is based on `SPLit` proposed in Chapter 1 and is orders of magnitude faster than the `SPLit` algorithm, which makes it applicable to Big Data problems such as data compression. `Twinning` can also be used for generating multiple splits of a given dataset to aid divide-and-conquer procedures, and k-fold cross validation.

In Chapter 3, we propose a novel framework for Gaussian process (GP) modeling. Contrary to the global, and local approximations proposed in the literature to address the computational bottleneck with exact GP modeling, we employ a combined global-local approach in building the approximation. Our framework uses a subset-of-data approach where the subset is a union of a set of global points designed to capture the global trend

in the data, and a set of local points specific to a given testing location to capture the local trend around the testing location. Here we use `Twinning`, presented in Chapter 2, to identify the set of global points. The correlation function is also modeled as a combination of a global, and a local kernel. The performance of our framework, which we refer to as `TwinGP`, is on par or better than the state-of-the-art GP modeling methods at a fraction of their computational cost.

CHAPTER 1

AN OPTIMAL METHOD FOR DATA SPLITTING

1.1 Introduction

For developing statistical and machine learning models, it is common to split the dataset into two parts: training and testing (Hastie et al., 2009; Stone, 1974). The training part is used for fitting the model, that is, to estimate the unknown parameters in the model. The model is then evaluated for its accuracy using the testing dataset. The reason for doing this is that if we were to use the entire dataset for fitting, the model would overfit the data and can lead to poor predictions in future scenarios. Therefore, holding out a portion of the dataset and testing the model for its performance before deploying it in the field can protect against unexpected issues that can arise due to overfitting.

In this chapter, we consider only datasets where each row is independent of other rows, that is, we will exclude cases such as time series data. The simplest and probably the most common strategy to split such a dataset is to randomly sample a fraction of the dataset. For example, 80% of the rows of the dataset can be randomly chosen for training and the remaining 20% can be used for testing. The aim of this chapter is to propose an optimal strategy to split the dataset.

Snee (1977) seems to be the first one who has carefully investigated several data splitting strategies. He proposed DUPLEX as the best strategy which was originally developed by Kennard as an improvement to another popular strategy CADEX (Kennard & Stone, 1969). Over the time, many other methods have been proposed in the literature for data splitting; see, for example the survey in Reitermanová (2010) and the comparative study in Xu and Goodacre (2018). Some of these methods will be discussed in the next section after proposing a mathematical formulation of the problem.

It is also common to hold out a portion of the training set for validation. The validation set can be used for fine-tuning the model performance such as for choosing hyper-parameters or regularization parameters in the model. In fact, the training set can be divided into multiple sets and the model can be trained using cross-validation. Our proposed method for optimally splitting the dataset into training and testing can also be used for these purposes by applying the method repeatedly on the training set.

The chapter is organized as follows. In Section 1.2, we provide a mathematical formulation of the problem and propose an optimal splitting method called `SPLIT` based on a technique for finding optimal representative points of a distribution known as *support points* (Mak & Joseph, 2018c). Support points are defined only for continuous variables. Therefore, we extend the support points methodology to deal with categorical variables in Section 1.3 so that `SPLIT` can be applied to both regression and classification problems. We apply `SPLIT` on several real datasets in Section 1.4 and compare its performance with random subsampling. Some concluding remarks are given in Section 1.5.

1.2 Methodology

Let $\mathbf{X} = (X_1, \dots, X_p)$ be the p input variables (or features) and Y the output variable. Let $\mathcal{D} = \{(\mathbf{X}_i, Y_i)\}_{i=1}^N$ be the dataset in hand. Our aim is to divide \mathcal{D} into two disjoint and mutually exclusive sets: \mathcal{D}^{train} and \mathcal{D}^{test} , where the training set \mathcal{D}^{train} contains N_{train} points and testing set \mathcal{D}^{test} contains N_{test} points with $N_{train} + N_{test} = N$.

1.2.1 Mathematical Formulation

Suppose the rows of the dataset are independent realizations from a distribution $F(\mathbf{X}, Y)$:

$$(\mathbf{X}_i, Y_i) \stackrel{iid}{\sim} F, \quad i = 1, \dots, N. \quad (1.1)$$

Let $g(\mathbf{X}; \boldsymbol{\theta})$ be the prediction model that we would like to fit to the data, where $\boldsymbol{\theta}$ is a set of unknown parameters in the model. The unknown parameters $\boldsymbol{\theta}$ will be estimated by minimizing a loss function $L(Y, g(\mathbf{X}; \boldsymbol{\theta}))$. Typical loss functions include squared or absolute error loss. More generally, the negative of the log-likelihood can be used as a loss function.

In postulating a prediction model $g(\mathbf{X}; \boldsymbol{\theta})$, our hope is that it will be close to the true model $E(Y|\mathbf{X})$ for some value of $\boldsymbol{\theta}$. However, the postulated model could be wrong and there may not exist a true value for $\boldsymbol{\theta}$. Thus it makes sense to try out different possible models on a training set and check their performance on the testing set so that we can identify the model that is closer to the truth.

The unknown parameters $\boldsymbol{\theta}$ can be estimated from the training set as

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{Argmin}} \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} L(Y_i^{\text{train}}, g(\mathbf{X}_i^{\text{train}}; \boldsymbol{\theta})), \quad (1.2)$$

which is a valid estimator provided that

$$(\mathbf{X}_i^{\text{train}}, Y_i^{\text{train}}) \sim F, \quad i = 1, \dots, N_{\text{train}}. \quad (1.3)$$

How should we split the dataset to obtain training and testing sets? We propose that the dataset should be split in such a way that the testing set gives an unbiased and efficient evaluation of the model's performance fitted using the training set.

To quantify the model's performance, define the generalization error as in Hastie et al. (2009, Ch. 7) by

$$\mathcal{E} = E_{\mathbf{X}, Y} \{L(Y, g(\mathbf{X}; \hat{\boldsymbol{\theta}})) | \mathcal{D}^{\text{train}}\}, \quad (1.4)$$

where the expectation is taken with respect to a realization (\mathbf{X}, Y) from F . Note that we do not include the randomness in $\hat{\boldsymbol{\theta}}$ induced by $\mathcal{D}^{\text{train}}$ for computing the expectation.

We can estimate \mathcal{E} if we have a sample of observations from F that is independent of the training set. We can use the testing set for this purpose. Thus, an estimate of \mathcal{E} can be

obtained as

$$\widehat{\mathcal{E}} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} L(Y_i^{test}, g(\mathbf{X}_i^{test}; \hat{\boldsymbol{\theta}})), \quad (1.5)$$

which works if

$$(\mathbf{X}_i^{test}, Y_i^{test}) \sim F, \quad i = 1, \dots, N_{test}. \quad (1.6)$$

A simple way to ensure condition (1.6) is to randomly sample N_{test} points from \mathcal{D} . Then, (1.5) can be viewed as the Monte Carlo (MC) estimate of \mathcal{E} . The question we are trying to answer is, if there is a better way to sample from \mathcal{D} so that we can get a more efficient estimate of \mathcal{E} . The answer to this question is affirmative. We can use Quasi-Monte Carlo (QMC) methods to improve the estimation of \mathcal{E} . It is well known that the error of MC estimates decreases at the rate $O(1/\sqrt{N_{test}})$, whereas when sampling from uniform distributions, the QMC error rate can be shown to be almost $O(1/N_{test})$ (Niederreiter, 1992). This is a substantial improvement in the error rate. However, most QMC methods focus on uniform distributions (Owen, 2013). Recently, Mak and Joseph (2018c) developed a method known as *support points* to obtain a QMC sample from general distributions. Although their theoretical results guarantee a convergence rate faster than MC by only a $\log N_{test}$ factor, much faster convergence rates are observed in practical implementations. This leads us to the proposed method `SPLit` (stands for Support Points-based split). We will discuss the method of support points and `SPLit` in detail after reviewing the existing data splitting methods in the next section.

1.2.2 Review of Data Splitting Methods

Interestingly, the original motivation behind `CADEX` (Kennard & Stone, 1969) and `DUPLEX` (Snee, 1977) was to create two sets with similar statistical properties, which agrees with the distributional condition mentioned in (1.6). However, these algorithms cannot achieve this objective. For example, consider a two-dimensional data generated using $(X_{1i}, X_{2i}) \stackrel{iid}{\sim} N_2(\mathbf{0}, \boldsymbol{\Sigma})$ for $i = 1, \dots, N$, where $\mathbf{0} = (0, 0)'$ and $\Sigma_{jk} = 0.5^{|j-k|}$. We will omit the response

here because these algorithms do not use it. Let $N = 1,000$ and $N_{test} = 100$. The CADEX and DUPLEX testing sets obtained using the R package `prospectr` (Stevens & Ramirez-Lopez, 2020) are shown in the left and middle top panels of Figure 1.1. We can see that both CADEX and DUPLEX testing points are too spread out and therefore, their distributions do not match with the distribution of the data. This can be seen more clearly in the marginal distributions shown in the bottom panels. For comparison, the testing set generated using the proposed `SPlit` method is shown in the top right panel. We can see that its distribution matches quite well with the distribution of the full data, as desired.

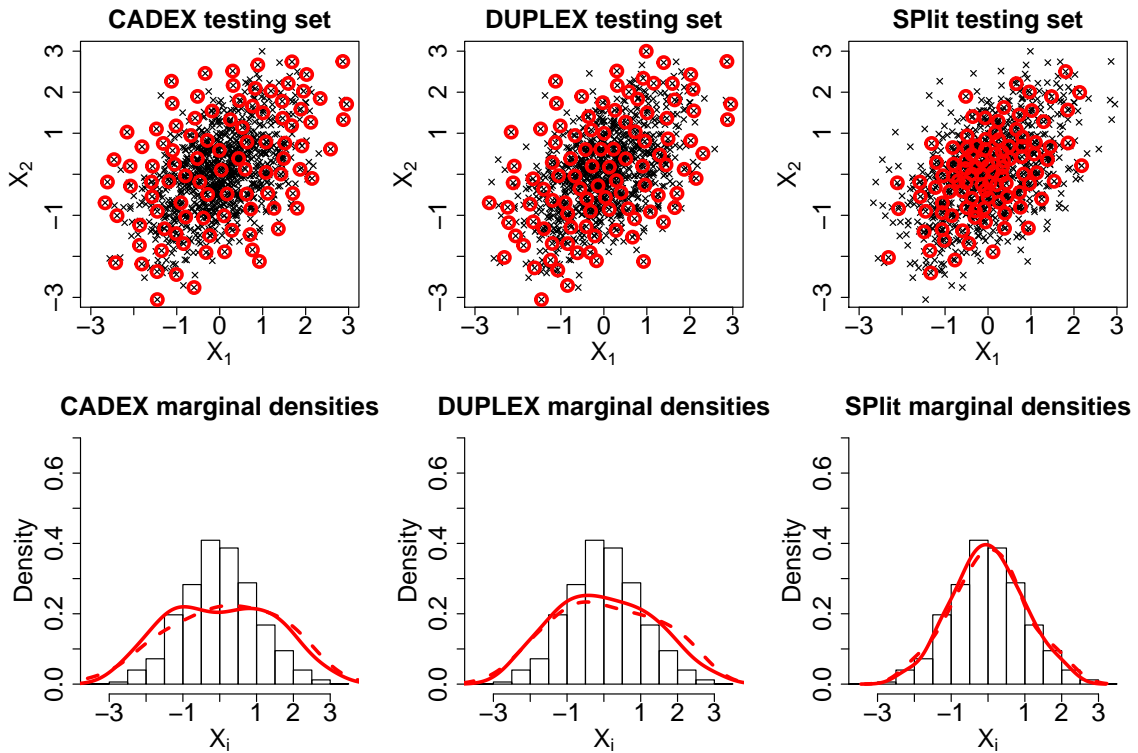


Figure 1.1: Comparison of CADEX and DUPLEX testing sets with `SPlit` testing set. 100 testing points (red circles) are chosen from 1,000 data points (black crosses). In the lower panels, the marginal densities of the testing sets are plotted over the histogram of the full data.

The sample set partitioning based on joint X-Y distances (SPXY) algorithm (Galvão et al., 2005) is a modification of the CADEX algorithm which incorporates the distances computed from the response values. Although incorporating Y was in the right direction of

(1.6), the algorithm suffers from the same issues of CADEX and DUPLEX algorithms. Bowden et al. (2002) proposed a data splitting method which uses global optimization techniques to match the mean and standard deviations of the testing set and the full data. This is again in the right direction of (1.6), however, matching the first two moments does not ensure distributional matching. May et al. (2010) proposed further improvement to the foregoing methodology using clustering-based stratified sampling. Although this provides an improvement, it is well-known that clustering distorts the original distribution (Zador, 1982) and hence cannot satisfy (1.6). In summary, none of existing data splitting methods except the random subsampling can ensure the distributional condition given in (1.6) or (1.3).

Before proceeding further, we would like to mention about another possible approach to solve the problem. One could think about splitting the dataset in such a way that the training set gives the best possible estimation of the model under a given loss function. For example, it is well-known that the best estimate of a linear regression model with linear effects of the predictors under the least squares criterion can be obtained by choosing the extreme values of the data from the predictor-space (Wang et al., 2019). Although such a choice can minimize the variance of the parameter estimates, the model may not perform well in the testing set if the original dataset is not generated from such a linear model. In our opinion, the testing set should provide a set of samples for an unbiased evaluation of the model performance and detect possible model deviations. For example, if we detect that quadratic terms are needed in the linear regression model, then having a training dataset with only extreme values of the predictors is not going to be useful. Therefore, the splitting method should be independent of the modeling choice and the loss function. Random subsampling achieves this aim and we will show that support points will achieve it even better!

Although support points have been used in the past for subsampling from big data (Mak & Joseph, 2018b), it was done for the purpose of saving storage space and time for fitting computationally expensive models due to limited resources. On the other hand, data

reduction is not the objective in our problem. After assessing the model’s performance using the testing set, the model will be re-estimated using the *full* data before deploying it for future predictions.

1.2.3 Support Points

Let $\mathbf{Z} = (\mathbf{X}, Y)$ be a vector of *continuous* variables. Then, the energy distance between the distribution $F(\mathbf{Z})$ and the empirical distribution of a set of points $\mathbf{z}_1, \dots, \mathbf{z}_n$ is defined as (Székely & Rizzo, 2013)

$$ED = \frac{2}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{z}_i - \mathbf{Z}\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2 - \mathbb{E} \|\mathbf{Z} - \mathbf{Z}'\|_2, \quad (1.7)$$

where $\mathbf{Z}, \mathbf{Z}' \sim F$, $\|\cdot\|_2$ is the Euclidean distance, and the expectations are taken with respect to F . Note that for the Euclidean distance to make sense, all the variables are standardized to have zero mean and unit variance. The energy distance will be small if the empirical distribution of $\mathbf{z}_1, \dots, \mathbf{z}_n$ is close to F . Therefore, Mak and Joseph (2018c) defined the support points of F as the minimizer of the energy distance:

$$\{\mathbf{z}_i^*\}_{i=1}^n \in \underset{\mathbf{z}_1, \dots, \mathbf{z}_n}{\text{Argmin}} ED = \underset{\mathbf{z}_1, \dots, \mathbf{z}_n}{\text{Argmin}} \left\{ \frac{2}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{z}_i - \mathbf{Z}\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2 \right\}. \quad (1.8)$$

They can be viewed as the representative points of the distribution F , which is the best set of n points to represent F according to the energy distance criterion. Mak and Joseph (2018c) showed that support points converge in distribution to F and therefore, they can be viewed as a QMC sample from F . This property makes support points different from other representative points of a distribution such as MSE-rep points (Fang & Wang, 1994) or principal points (Flury, 1990) which do not possess distributional convergence (Zador, 1982).

In our problem, we do not have F . Instead we only have a dataset \mathcal{D} , which is a set of independent realizations from F . Therefore, to compute the support points, we can replace

the expectation in (1.8) with a Monte Carlo average computed over \mathcal{D} :

$$\{\mathbf{z}_i^*\}_{i=1}^n \in \underset{\mathbf{z}_1, \dots, \mathbf{z}_n}{\text{Argmin}} \left\{ \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{z}_i - \mathbf{Z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2 \right\}. \quad (1.9)$$

This simple extension to real data settings is another advantage of support points, which cannot be done for other representative points such as minimum energy design (Joseph et al., 2015) and Stein points (W. Y. Chen et al., 2018) even though they possess distributional convergence.

At first sight, the optimization in (1.9) appears to be a very hard problem. The objective function is nonlinear and non-convex. Moreover, the number of variables in the optimization is $n(p + 1)$, which can be extremely high even for small datasets. However, the objective function has a nice feature; it is a difference of two convex functions. By exploiting this feature, Mak and Joseph (2018c) developed an efficient algorithm based on difference-of-convex programming techniques, which can be used for quickly finding the support points. Although a global optimum is not guaranteed, an approximate solution can be obtained in a reasonable amount of time. Their algorithm is implemented in the R package `support` (Mak, 2019). Although it is possible to create representative points using other goodness-of-fit test statistics (Hickernell, 1999) and kernel functions (Y. Chen et al., 2010) instead of the energy distance criterion, they do not seem to possess the computational advantage and robustness of support points.

1.2.4 SPlit

We can use the support points obtained from (1.9) as the testing set with $n = N_{test}$ and the remaining data can be used as the training set. Alternatively, we can use support points to obtain the training set with $n = N_{train}$ and then use the remaining data as the testing set. However, the computational complexity of the algorithm used for generating support points is $\mathcal{O}(n^2(p + 1))$. Since N_{test} is usually smaller than N_{train} , it will be faster

to generate the testing set using support points than the training set. In general, we let $n = \min\{N_{test}, N_{train}\} = \min\{N\gamma, N(1 - \gamma)\}$, where $\gamma = N_{test}/N$ is the splitting ratio.

As discussed earlier, the testing set generated using support points is expected to work better than a random sample from \mathcal{D} . However, there is one drawback. Support points need not be a subsample of the original dataset. This is because the optimization in (1.9) is done on a continuous space and therefore, the optimal solution need not be part of \mathcal{D} . To get a subsample, we actually need to solve the following *discrete optimization* problem:

$$\{\mathbf{z}_i^*\}_{i=1}^n \in \underset{\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathcal{D}}{\text{Argmin}} \left\{ \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{z}_i - \mathbf{z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2 \right\}. \quad (1.10)$$

Our initial attempts to solve this problem using state-of-the-art integer programming techniques showed that they are accurate, but too slow in finding the optimal solution. Computational speed is crucial for our method to succeed because otherwise it will not be attractive against the computationally cheap alternative of random subsampling. Therefore, here we propose an approximate but efficient algorithm to subsample from \mathcal{D} .

We will first find the support points in a continuous space as in (1.9), which is very fast. We will then choose the closest points in \mathcal{D} to $\{\mathbf{z}_i^*\}_{i=1}^n$ according to the Euclidean distance. This can be done efficiently even for big datasets using KD-Tree based nearest neighbor algorithms. However, a naive nearest neighbor assignment can lead to duplicates and therefore, the remaining data points can become more than $N - n$. Moreover, separating the points can increase the second term in (1.10) and thus potentially improve the energy distance criterion. This can be achieved by doing the nearest neighbor assignment sequentially. Our method is summarized in Algorithm 1 and is implemented in the R package `SPLIT`. A critical step in this algorithm is to update the KD-Tree efficiently when a point is removed from the dataset. We use `nanoflann`, a C++ header-only library (Blanco & Rai, 2014), which allows for lazy deletion of a data point from the KD-Tree without having to rebuild the KD-Tree every time a point is removed from the dataset.

Algorithm 1 SPlit: Splitting a dataset \mathcal{D} with splitting ratio γ [R package: SPlit]

```
1: Input  $\mathcal{D} \in \mathbb{R}^{N \times (p+1)}$  and  $\gamma = N_{test}/N$ 
2: Standardize the columns of  $\mathcal{D}$ 
3:  $n \leftarrow \min\{N\gamma, N(1 - \gamma)\}$ 
4: Compute  $\{\mathbf{z}_i^*\}_{i=1}^n$  using (1.9)
5:  $\mathcal{D}_1 \leftarrow \{\}$ 
6: for  $i \in \{1, \dots, n\}$  do
7:    $\hat{\mathbf{u}} \in \arg \min_{\mathbf{u}} \{\|\mathbf{u} - \mathbf{z}_i^*\|_2 : \mathbf{u} \in \mathcal{D}\}$ 
8:    $\mathcal{D}_1 \leftarrow \mathcal{D}_1 \cup \{\hat{\mathbf{u}}\}$ 
9:    $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\hat{\mathbf{u}}\}$ 
10: end for
11:  $\mathcal{D}_2 \leftarrow \mathcal{D}$ 
12: return  $\mathcal{D}_1, \mathcal{D}_2$ 
```

1.2.5 Visualization

Consider a simple example for visualization purposes. Suppose we generate $N = 100$ points as follows: $X_i \stackrel{iid}{\sim} N(0, 1)$ and $Y_i|X_i \stackrel{iid}{\sim} N(X_i^2, 1)$ for $i = 1, \dots, N$. Both X and Y values are standardized to have zero mean and unit variance. Figure 1.2 shows the optimal testing set obtained using SPlit and a random testing set obtained using random subsampling without replacement. We can see that the points in the SPlit testing set are well spread out throughout the region and provide a much better point set to evaluate the model performance than the random testing set.

Most statistical and machine learning models have some hyper-parameters or regularization parameters, which are commonly estimated from the training set by holding out a validation set, say of size N_{valid} . One simple approach to create an optimal validation set is to apply the SPlit algorithm on the training set. However, it may happen that such a set is close to the points in the testing set, which is not good as it may lead to a biased testing performance. We want the validation points to stay away from the testing points so that the testing performance is not influenced by the model estimation/validation step. This can be achieved as follows. Let $\{\mathbf{z}_1, \dots, \mathbf{z}_{N_{test}}\}$ be the testing set and $\{\mathbf{z}_{N_{test}+1}, \dots, \mathbf{z}_n\}$ the validation

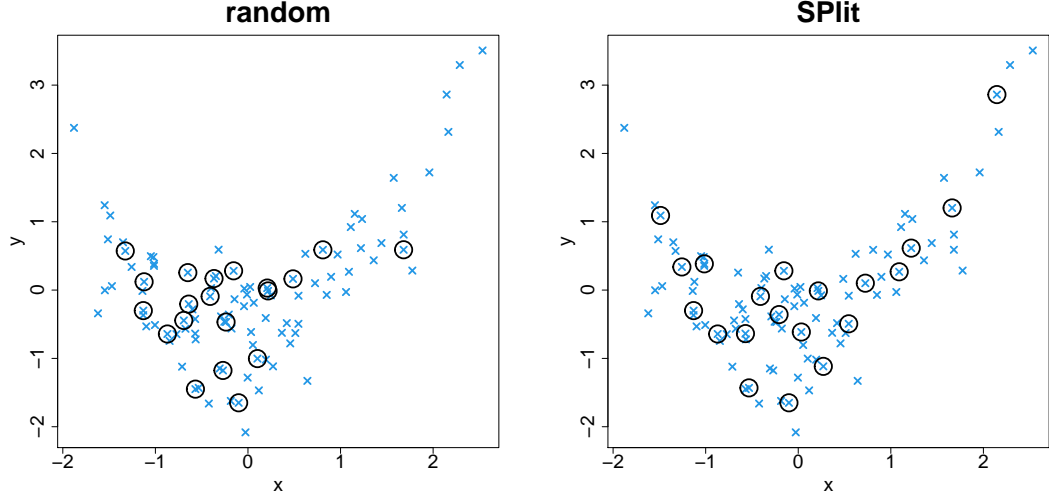


Figure 1.2: The circles are the testing set obtained using random (left) and SPlit (right) subsampling.

set, where $n = N_{test} + N_{valid}$. Then, the optimal validation points can be obtained as

$$\{\mathbf{z}_i^*\}_{i=N_{test}+1}^n \in \underset{\mathbf{z}_{N_{test}+1}, \dots, \mathbf{z}_n \in \mathcal{D}}}{\text{Argmin}} \left\{ \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{z}_i - \mathbf{z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2 \right\}, \quad (1.11)$$

where the optimization takes place only over the validation points with the testing points fixed at $\{\mathbf{z}_1^*, \dots, \mathbf{z}_{N_{test}}^*\}$. Because of the second term in the energy distance criterion, the validation points will move away from the testing points.

Figure 1.3 shows 20 points selected out of the 80 training points using (1.11). A random subsample is also shown in the same figure for comparison. Clearly, the optimal validation set created using our method is a much better representative set of the original dataset and therefore, it can do a much better job in tuning the hyper-parameter or regularization parameter than using a random validation set. In fact, we can sequentially divide the training set into K sets by repeated application of this method and use them for K -fold cross-validation. Because of the importance of this problem, we will leave this topic for future research.

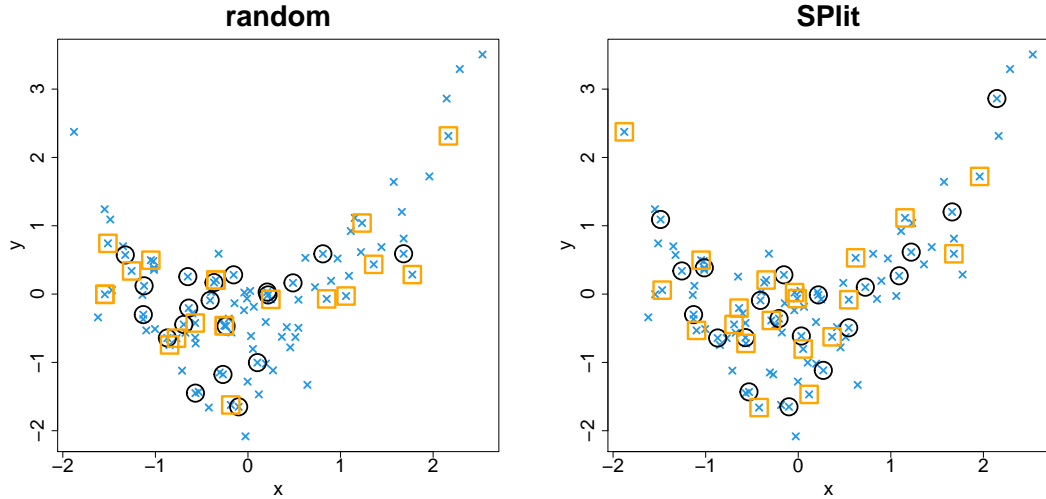


Figure 1.3: The squares are the validation set obtained using random (left) and SPlit (right) subsampling from the training set. The testing set is shown as circles.

1.2.6 Simulations

Since support points create a dependent set, one may wonder if the testing set and training set are related and if the dependence will create a bias in the estimation of the generalization error in (1.5). We will perform some simulations to check this. Consider again the data generating model discussed in the previous section:

$$Y_i = X_i^2 + \epsilon_i, \quad (1.12)$$

where $\epsilon_i \stackrel{iid}{\sim} N(0, 1)$ and $X_i \stackrel{iid}{\sim} N(0, 1)$, for $i = 1, \dots, N$. Let $N = 1,000$. Suppose we fit the following r th degree polynomial model to the data:

$$Y_i = g(X_i; \boldsymbol{\theta}) + \epsilon_i,$$

where $g(X; \boldsymbol{\theta}) = \theta_0 + \theta_1 X + \theta_2 X^2 + \dots + \theta_r X^r$ and $\epsilon_i \stackrel{iid}{\sim} N(0, \sigma^2)$. The unknown parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_r)'$ can be estimated from the training set using least squares. The generalization

error can then be computed as

$$\begin{aligned}
\mathcal{E} &= E_{X,Y} \left[\{Y - g(X; \hat{\theta})\}^2 | \mathcal{D}^{train} \right] \\
&= E_{X,Y} \left[\{Y - g(X; \hat{\theta})\}^2 \right] \quad (\text{by independence}) \\
&= E_X \left(E_{Y|X} \left[\{Y - g(X; \hat{\theta})\}^2 | X \right] \right) \\
&= E_X \left(\{X^2 - g(X; \hat{\theta})\}^2 \right) + 1.
\end{aligned}$$

We can divide a given dataset into training and testing sets using various data splitting methods and estimate the generalization error using (1.5). Thus, we can compute the estimation error of a data splitting method as $\hat{\mathcal{E}} - \mathcal{E}$. For comparison, we use `SPLIT`, random subsampling, `CADEX`, and `DUPLEX`. This procedure is repeated 100 times by generating testing sets with splitting ratios of 10% and 50%. Owing to the deterministic nature, `CADEX` and `DUPLEX` produce the same testing set each time. On the other hand, some variability is observed in the testing sets produced by `SPLIT`, which is mainly due to the random initialization and convergence to local optima of the support points' algorithm. Figure 1.4 shows the estimation errors over different values of r .

We can see that the bias in the estimation of generalization error using `SPLIT` is small compared to the other data splitting methods. This confirms the validity of the proposed method.

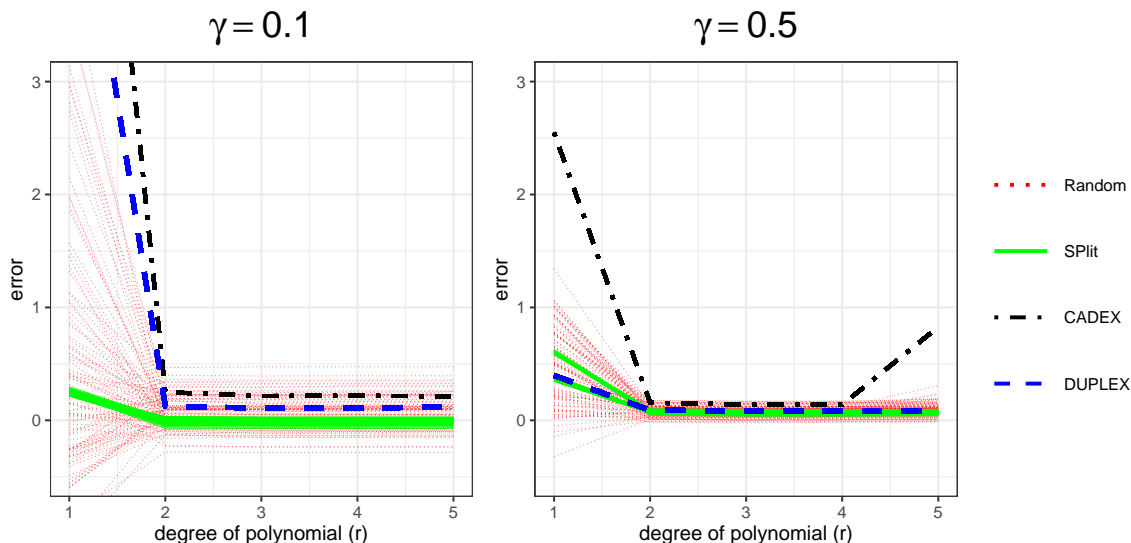


Figure 1.4: The error in the estimation of generalization error with splitting ratios of 10% (left) and 50% (right) for different data splitting methods over 100 iterations.

1.3 Categorical Variables

Energy distance in (1.7) is defined only for continuous variables because its definition involves Euclidean distances and therefore, support points can only be found for datasets with continuous variables. However, a dataset can have categorical predictors and/or responses. Therefore, it is important to extend the support points methodology to deal with categorical variables in order to implement SPlit.

For simplicity of notations, let us consider the case of only a single categorical variable, which could be a predictor or a response. It is easy to extend our methodology to multiple categorical variables, which will be explained later. Let m be the number of levels of the categorical variable and N_i be the corresponding number of points in the dataset at the i th level, $i = 1, \dots, m$.

The most naive approach to deal with a categorical variable is to simply ignore it and find the n support points from a dataset containing N points using only the continuous variables as in (1.9). For illustration, consider the same example used in Section 2.4, except that we generate the data as follows. Let $X_{1i} \stackrel{iid}{\sim} N(0, 1)$ and $X_{2i}|X_{1i} \stackrel{iid}{\sim} N(X_{1i}^2, 1)$ for

$i = 1, \dots, N$ with $N = 100$. They are scaled to have zero mean and unit variance. Consider a nominal categorical response variable with three levels: Red, Green, and Blue. The points are classified as Red with probability $\Phi(-6X_1 - X_2 - 5)$ and Blue with probability $\Phi(6X_1 - X_2 - 5)$, where $\Phi(\cdot)$ is the standard normal distribution function. The remaining points are classified as Green. The data are shown in Figure 1.5.

Suppose our aim is to generate a testing set of 20 points. The result of applying the naive method is shown in the left panel of Figure 1.5. In this dataset there are $N_1 = 21$ Red, $N_2 = 59$ Green, and $N_3 = 20$ Blue. A testing set gives a good representation of the categorical variable if

$$\frac{n_i}{n} \approx \frac{N_i}{N} \text{ for all } i = 1, \dots, m, \quad (1.13)$$

where n_i is the number of testing samples for the i th level. Thus, we should have approximately four Red, 12 Green, and four Blue in the testing set. However, the naive method gives only two Red and three Blue, which is quite disproportionate to the number of Red and Blue in the dataset. This is expected because the naive method does not use the information in the categorical response for splitting.

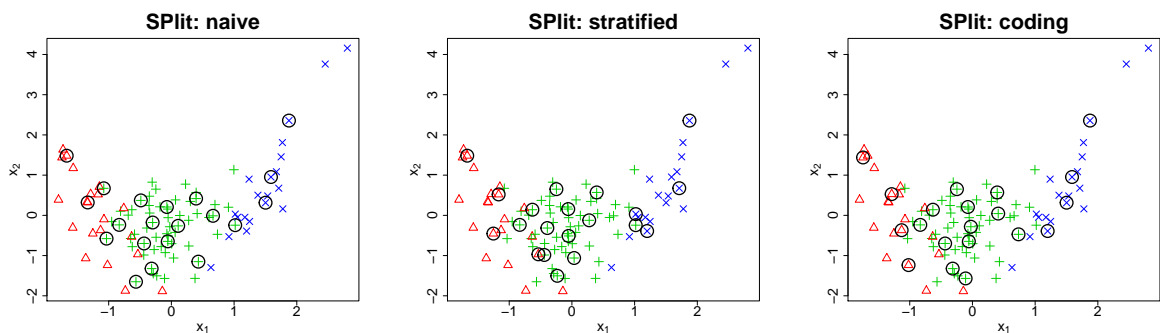


Figure 1.5: The circles are the testing set obtained using naive method (left), stratified proportional method (center), and the proposed coding-based method (right). The three categorical levels are shown as red triangles, green pluses, and blue crosses.

Another possible approach that can ensure the proportional sampling in (1.13) is to first choose $n_i \approx N_i/Nn$ and then find n_i support points from the dataset containing only the i th

level of the categorical variable while ignoring the remaining part of the dataset. The results of this stratified proportional method is shown in the middle panel of Figure 1.5. We can see that some points are too close and almost overlapping. Thus, although the stratified proportional sampling approach ensures perfect balancing of categorical levels, the support points in the continuous space may not be representative.

A yet another approach to deal with categorical variables is to first convert them into numerical variables and then use the methodology that we developed earlier for continuous variables. The first step is to represent the m levels of the categorical variable using $m - 1$ dummy variables assuming that the model has a parameter to represent the mean of the data. There are many choices for creating the dummy variables such as treatment coding, Helmert coding, sum coding, orthogonal polynomial coding, etc. (Faraway, 2015, ch.14). Here we use Helmert coding as we have observed better numerical stability with it in the support points' algorithm.

Consider the example again. Using Helmert coding, the Red, Green, and Blue levels are represented using two dummy variables $d_1 = (-1, 1, 0)$ and $d_2 = (-1, -1, 2)$. Now the `SPlit` algorithm can be applied after standardizing the four columns of the augmented dataset. The result is shown in the last panel of Figure 1.5. We can see that the categorical levels are well-balanced and the points are well-spread out in the continuous space. Thus, this coding approach seems to work very well. Moreover, it can be easily adapted for ordinal categorical variables through scoring (Wu & Hamada, 2011, p. 647). Furthermore, it can be used for multiple categorical variables by coding each variable separately. Thus, this approach appears to be very general and simple to implement and therefore, we will adopt it in the `SPlit` algorithm to handle categorical variables.

1.4 Examples

In this section we will compare `SPlit` with random subsampling on real datasets for both regression and classification problems.

1.4.1 Regression

Consider the concrete compressive strength dataset from Yeh (1998) which can be obtained from the UCI Machine Learning Repository (Dua & Graff, 2017). This dataset has eight continuous predictors pertaining to the concrete’s ingredients and age. The response is the concrete’s compressive strength. We will make an 80-20 split of this dataset which has 1,030 rows. Thus $N_{train} = 824$ and $N_{test} = 206$. The split is done using both `SPlit` and random subsampling. All the nine variables are normalized to mean 0 and standard deviation 1 before splitting.

A good splitting procedure should work well for all possible modeling choices. Therefore, to check the robustness against different modeling choices, we choose a linear regression model with linear main effects estimated using LASSO (Tibshirani, 1996) and a nonlinear-nonparametric regression model estimated using random forest (Breiman, 2001). Both the models are fitted on the training set using the default settings of the R packages `glmnet` (J. Friedman et al., 2010) and `randomForest` (Liaw & Wiener, 2002). Then we compute the root mean squared prediction error (RMSE) on the testing set to evaluate the models’ prediction performance. We repeat this procedure 500 times, where the same split is used for fitting both the LASSO and random forest.

The testing RMSE values for the 500 simulations are shown in Figure 1.6. We can see that on the average the testing RMSE is lower for `SPlit` compared to random subsampling. This improvement is much larger for random forest compared to LASSO. We also note significant improvement in the worst-case performance of `SPlit` over random subsampling. Furthermore, the variability in the testing RMSE is much smaller for `SPlit` compared to random subsampling and therefore, a more consistent conclusion can be drawn using `SPlit`. Thus, the simulation clearly shows that `SPlit` produces testing and training set that are much better for model fitting and evaluation. Computation of `SPlit` for this dataset took on an average 1.6 seconds on a computer with 6-core 2.6 GHz Intel processor, which is a negligibly small price that we need to pay for the improved performance over random

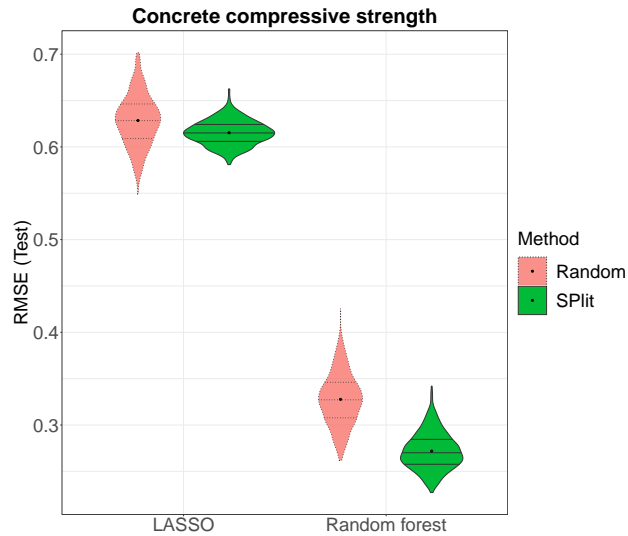


Figure 1.6: Distribution of root mean squared error (RMSE) over 500 random and SPlit subsampling splits for the concrete compressive strength dataset.

subsampling.

Dataset	Size	Predictors	Response
Abalone	4177×9	7 continuous 1 categorical (3 levels)	Continuous
Airfoil self-noise	1503×6	5 continuous	Continuous
Meat spectroscopy	215×101	100 continuous	Continuous
Philadelphia birthweights	1115×5	2 continuous 2 categorical (2 levels each)	Continuous

Table 1.1: Description of datasets considered for regression.

We repeated the simulation with several other datasets, namely Abalone (Nash et al., 1994), Airfoil self-noise (Brooks et al., 1989), Meat spectroscopy (Thodberg, 1993), and Philadelphia birthweights (Elo et al., 2001). Abalone and Airfoil self-noise datasets can be obtained from the UCI Machine Learning Repository (Dua & Graff, 2017), while Meat spectroscopy and Philadelphia birthweights can be obtained from the faraway (Faraway, 2015) package in R. The details of these datasets are summarized in Table 1.1. Figure 1.7 shows the testing RMSE values for both LASSO and random forest. We see similar trends as before on all the datasets; SPlit gives a better testing performance on the average than random subsampling and a substantial improvement in the worst-case testing performance.

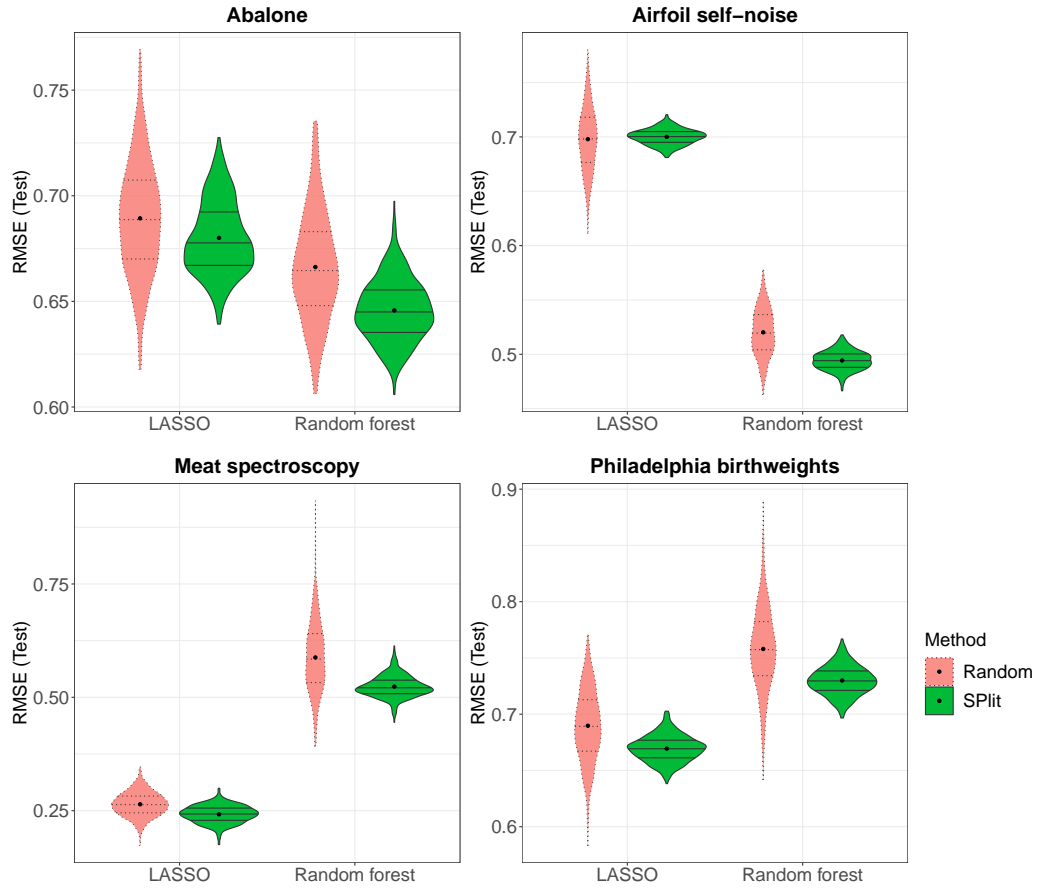


Figure 1.7: Distribution of root mean squared error (RMSE) over 500 random and SPlit subsampling splits for the datasets described in Table 1.1.

1.4.2 Classification

For checking the performance of SPlit on classification problems, consider the famous Iris dataset (Fisher, 1936). The Iris dataset has four continuous predictors (sepal length, sepal width, petal length, and petal width) and a categorical response with three levels representing the three types of Iris flowers (setosa, versicolor, and virginica). There are 150 rows in total with 50 rows for each flower type. Following the discussion in Section 3, the flower type is converted into two continuous dummy variables using Helmert coding. Thus, the resulting dataset has six continuous columns. For modeling we will use multinomial logistic regression and random forest. The classification performance will be assessed using the

residual deviance (D) defined as

$$D := 2 \sum_{i=1}^I \sum_{j=1}^J y_{ij} \cdot \ln\left(\frac{y_{ij}}{\hat{p}_{ij}}\right), \quad (1.14)$$

where I is the number of rows, J the number of classes, $y_{ij} \in \{0, 1\}$ is 1 if row i corresponds to class j and 0 otherwise, and \hat{p}_{ij} is the probability that row i belongs to class j as predicted by the model. Note that $0 \log 0$ is taken as 0 by definition.

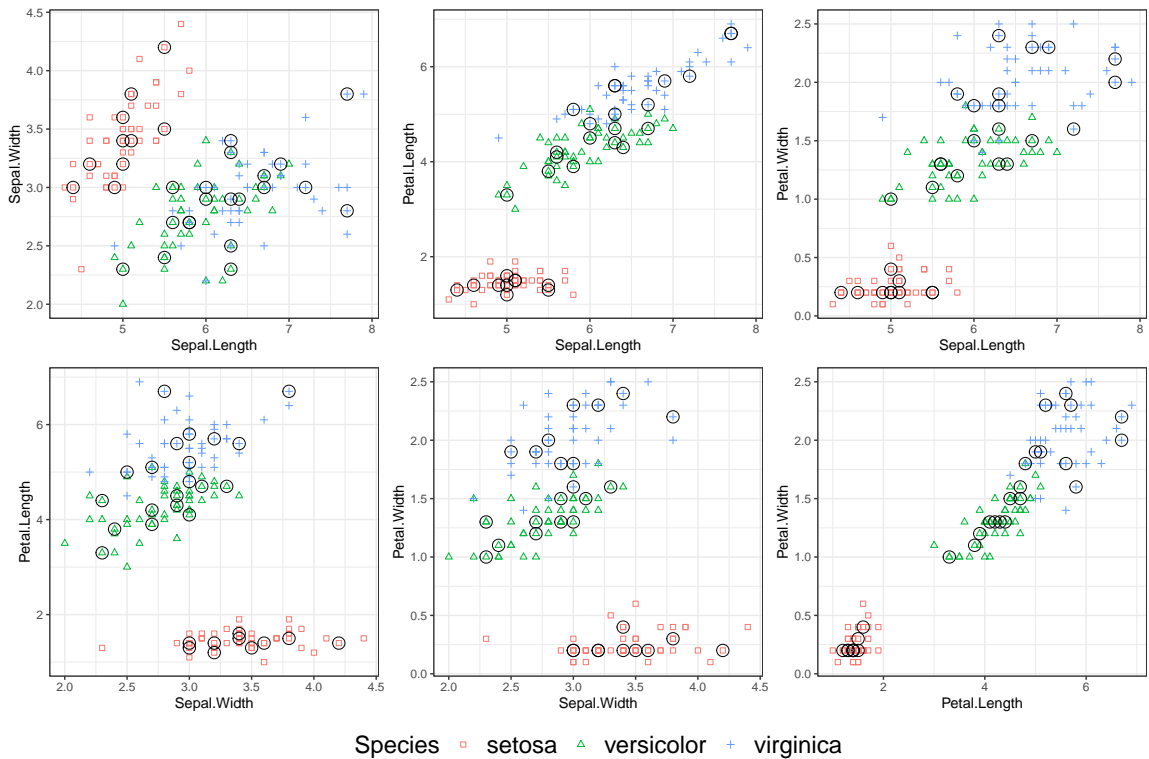


Figure 1.8: Visualizing a SPlit subsampling testing set (circles) for the Iris dataset.

Figure 1.8 shows a testing set selected by SPlit. We can see that they are well-balanced among the three classes and the points are well-spread out in the space of the four continuous predictors. We fit multinomial logistic regression and random forest on the training set and then the residual deviance is computed on the testing set. This is then repeated 500 times. Figure 1.9 shows the deviance results for SPlit, random, and stratified proportional random subsampling. We can see that again SPlit gives significantly better average and worst-case

performance compared to both random and stratified proportional random subsampling.

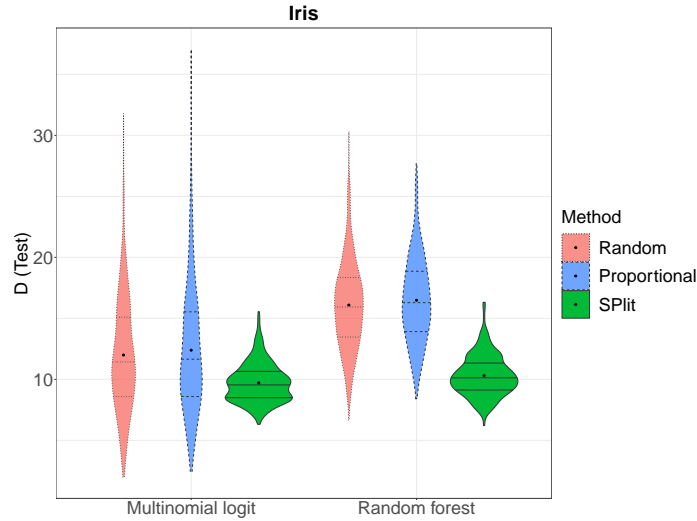


Figure 1.9: Distribution of residual deviance (D) over 500 random, stratified proportional random, and SPlit subsampling splits for the Iris dataset.

Dataset	Size	Predictors	Response
Banknote authentication	1372×5	4 continuous	Categorical (2 levels)
Breast cancer (<i>diagnostic, Wisconsin</i>)	569×31	30 continuous	Categorical (2 levels)
Cardiotocography	2126×22	20 continuous 1 categorical (3 levels)	Categorical (3 levels)
Glass identification	214×10	9 continuous	Categorical (6 levels)

Table 1.2: Description of datasets considered for classification.

The foregoing study is repeated for four other datasets: Banknote authentication, Breast cancer (*diagnostic, Wisconsin*) (Street et al., 1993), Cardiotocography (Ayres-de-Campos et al., 2000), and Glass identification (Evet & Spiehler, 1989), all of which can be obtained from the UCI Machine Learning Repository (Dua & Graff, 2017). The details of these datasets are summarized in Table 1.2 and the results on the residual deviance are shown in Figure 1.10. It is possible to encounter ∞ while calculating deviance; for the purpose of plotting, ∞ is replaced with the maximum finite deviance obtained from the remainder of the 500 simulations. We can see that SPlit gives a better performance than both random and stratified proportional random subsampling in all the cases. The improvement realized

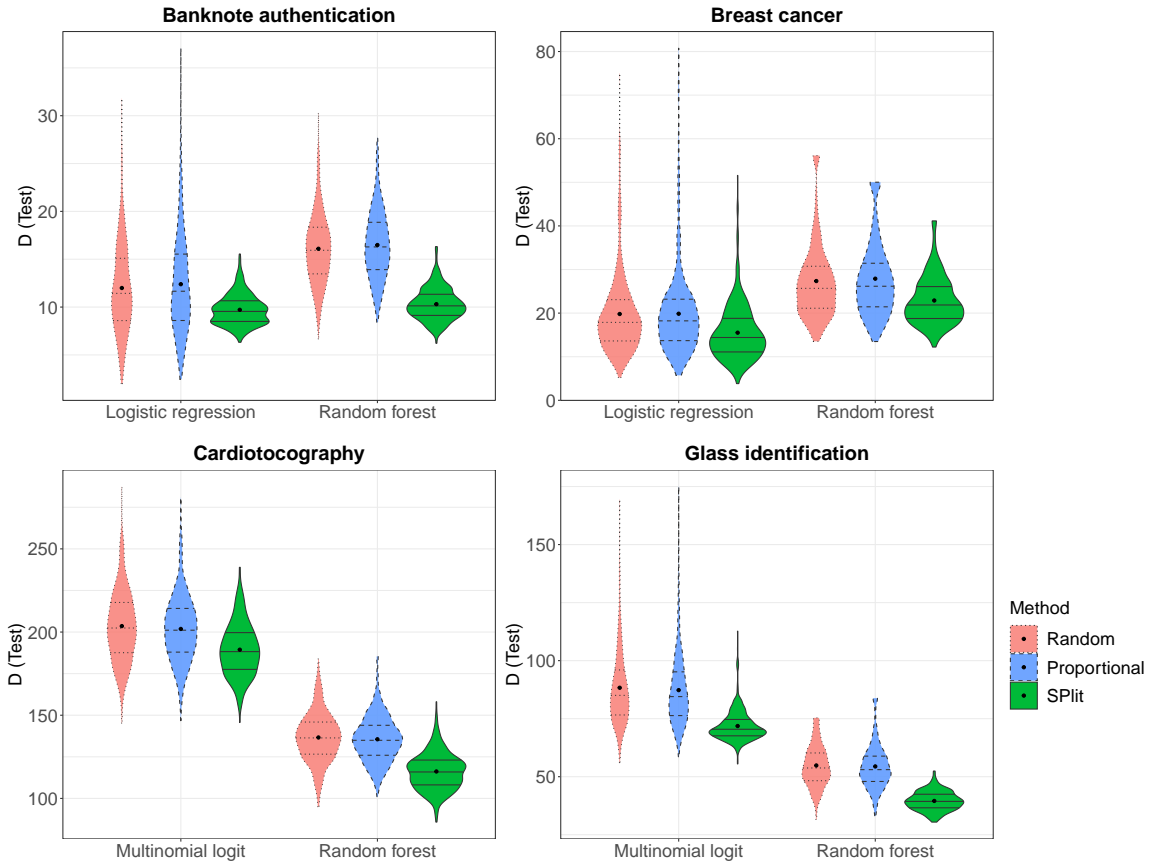


Figure 1.10: Distribution of residual deviance (D) over 500 random, stratified proportional random, and SPlit subsampling splits for the datasets described in Table 1.2.

varies over the datasets and modeling methods, but SPlit has a clear advantage over both random and stratified proportional random subsampling.

1.5 Conclusions

Random subsampling is probably the most widely used method for splitting a dataset for testing and training. In this chapter, we have proposed a new method called SPlit for optimally splitting the dataset. It is done by first finding support points of the dataset and then using an efficient nearest neighbor algorithm to choose the subsamples. They are then used as the testing set and the remaining as the training set. The support points give the best possible representation of the dataset (according to the energy distance criterion) and therefore, SPlit is expected to produce a testing set that is best for evaluating the

performance of a model fitted on the training set. The ability of support points to match the distribution of the full data is one of its big advantage over the other deterministic data splitting methods such as CADEX and DUPLEX. We have also extended the method of support points to deal with categorical variables. Thus, SPlit can be applied to both regression and classification problems. We have also briefly discussed on how a sequential application of the support points can be used to generate validation and cross-validation sets, but further development on this topic is left for future research.

We have applied SPlit on several datasets for both regression and classification using different choices of modeling methods and found that SPlit improves the average testing performance in almost all the cases with substantial improvement in the worst-case predictions. The variability in the testing performance metric using SPlit is found to be much smaller than that of random subsampling, which shows that the results and the findings of a statistical study would be much more reproducible if we were to use SPlit.

CHAPTER 2

DATA TWINNING

2.1 Introduction

Often in statistics and machine learning we are required to partition a dataset, e.g., when (i) splitting a dataset for training and testing, (ii) subsampling from Big Data for conducting tractable statistical analysis or to save storage space, (iii) generating multiple splits of a dataset for divide-and-conquer procedures to act upon, and (iv) creating k -fold cross validation sets for model tuning and validation. For this purpose, we propose a novel method named **Twinning** that can be used for partitioning a dataset into statistically similar sets.

Twinning is motivated from the work on optimal data splitting for model validation, presented in Chapter 1. For model validation, the common practice is to randomly split the dataset into training and testing sets, e.g., for an 80-20 split, 20% of the dataset is selected randomly for testing, while the remaining 80% is used for training the model. It is easy to see that such random splitting can plausibly give rise to pathological splits, wherein the training and testing sets cover roughly disjoint regions of the feature space, thereby resulting in poor testing performance of the model. Clearly there is a need to systematically split data such that the training set provides sufficient coverage of the feature space. CADEX by Kennard and Stone (1969) marks the beginning of such data splitting procedures in the literature. CADEX pushes majority of the extreme points in the feature space into the training set, while in DUPLEX, a procedure later presented by Snee (1977) as an improvement over CADEX, the extreme points are about equally distributed between the training and testing sets, thereby producing a more robust testing set for validation. Reitermanová (2010) provides a detailed survey of various data splitting procedures that are geared towards this goal.

If rows of a given dataset, including the response, are assumed to be independent

realizations from a distribution \mathcal{F} , then an unbiased estimate of the model’s generalization error is obtained when the testing set itself is a realization of \mathcal{F} . Random data splitting achieves this distributional similarity, but not CADEX and DUPLEX; furthermore, CADEX and DUPLEX do not include the response in their computations. The SPXY procedure proposed by Galvão et al. (2005) is similar to CADEX, but includes the response, even still, the distributional similarity is lacking.

The procedure `SPLIT` produces testing sets statistically similar to the original dataset. Both parametric and non-parametric models built using `SPLIT` exhibit promising results over random data splitting, i.e., improved and consistent testing performance; only caveat being the computational complexity of making the split. Consider some d -dimensional datasets generated by sampling from a multivariate normal with $\boldsymbol{\mu} = [0, \dots, 0]^\top \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d} : \Sigma_{ij} = 0.5^{|i-j|}, \forall i, j \in \{1, \dots, d\}$. Figure 2.1 plots the computation time to make an 80-20 split using 500 iterations of the original algorithm for `SPLIT` on a 36-core Intel 3.0 GHz processor, against the size of the dataset. Although the algorithm can be executed in parallel, the quadratic growth suggests a two day wait to split the 16-dimensional dataset if it had a million rows, even with access to 36 cores. Hence, though apt for model validation, the computational complexity of `SPLIT` holds it back from being applied to Big Data that are prevalent today in every domain. In this work, we develop an efficient algorithm named `Twinning` that is capable of splitting Big Data with the same objective as `SPLIT`. As will be shown, the computational efficiency of `Twinning`, coupled with the statistical properties of the splits generated, broadens its scope to a wide variety of problems that are inaccessible for `SPLIT`.

The remainder of the chapter is organized as follows. Section 2.2 reviews the theory behind `SPLIT`. Section 2.3 presents the new `Twinning` algorithm, and Section 2.4 provides several computational experiments to assess the performance of `Twinning`. Section 2.5 extends `Twinning` for generating multiple splits of the dataset. Section 2.6 discusses applications of `Twinning` in data splitting, data compression, and cross-validation. Finally,

we state our concluding remarks in Section 2.7.

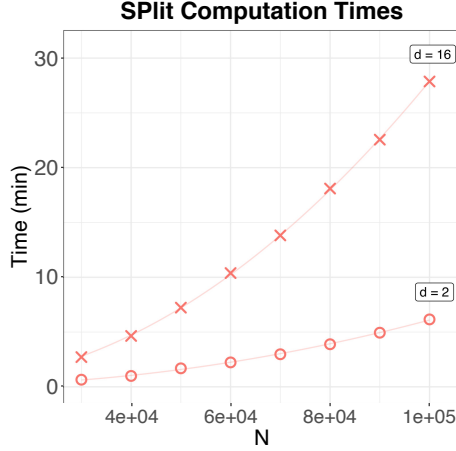


Figure 2.1: Computation times for making an 80-20 split of d -dimensional multivariate normal datasets with N rows, using 500 iterations of the original algorithm for `SPlit`, on a 36-core processor.

2.2 Review of `SPlit`

Let $\mathcal{D} = \{\mathbf{Z}_i = [\mathbf{X}_i, Y_i]\}_{i=1}^N \in \mathbb{R}^{N \times d}$ be the given dataset, where \mathbf{X}_i is a $d-1$ dimensional vector representing the $d-1$ features in the i^{th} row, and Y_i denotes the corresponding response value. Assume that each row of the dataset is independently drawn from a distribution \mathcal{F} :

$$(\mathbf{X}_i, Y_i) \stackrel{iid}{\sim} \mathcal{F}, \quad i = 1, \dots, N.$$

The aim is to fit a model $g(\mathbf{X}; \boldsymbol{\theta})$ to the data, where $\boldsymbol{\theta}$ is the unknown set of parameters in the model. To protect against overfitting, we will not use the entire data for estimation. Instead, the dataset is split into two sets: testing set (\mathcal{D}^1) and training set (\mathcal{D}^2) with sizes n and $N-n$, respectively. Then, \mathcal{D}^2 will be used for parameter estimation, and \mathcal{D}^1 will be used for assessing the model performance.

To quantify the model's performance, define the generalization error as in Hastie et al.

(2009, Ch. 7) by

$$\mathcal{E} = E_{\mathbf{X}, Y} \{L(Y, g(\mathbf{X}; \hat{\boldsymbol{\theta}})) | \mathcal{D}^2\}, \quad (2.1)$$

where $\hat{\boldsymbol{\theta}}$ is the estimate of $\boldsymbol{\theta}$ obtained from the training set by minimizing a loss $L(Y, g(\mathbf{X}; \boldsymbol{\theta}))$.

An estimate of the generalization error can be obtained from the testing set as

$$\widehat{\mathcal{E}} = \frac{1}{n} \sum_{i=1}^n L(Y_i^1, g(\mathbf{X}_i^1; \hat{\boldsymbol{\theta}})), \quad (2.2)$$

where $\mathbf{U}_i = (\mathbf{X}_i^1, Y_i^1)$ denote the i^{th} sample in \mathcal{D}^1 . The estimate of the generalization error will be unbiased if

$$\mathbf{U}_i \sim \mathcal{F}, \quad i = 1, \dots, n. \quad (2.3)$$

Random sampling is the easiest method to ensure condition (2.3). However, random sampling gives a high error rate of $O(n^{-1/2})$ for $|\widehat{\mathcal{E}} - \mathcal{E}|$. Mak and Joseph (2018c) showed that, under some regularity conditions,

$$|\widehat{\mathcal{E}} - \mathcal{E}| \leq C \sqrt{\mathbb{E}\mathbb{D}}, \quad (2.4)$$

where C is a constant that does not depend on the testing sample, and $\mathbb{E}\mathbb{D}$ is the energy distance (Székely & Rizzo, 2013) between the testing sample and the distribution \mathcal{F} . The energy distance can be estimated from the data using

$$\overline{\mathbb{E}\mathbb{D}}_{n, N} := \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{U}_i - \mathbf{Z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 - \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N \|\mathbf{Z}_i - \mathbf{Z}_j\|_2,$$

where $\|\cdot\|_2$ is the ℓ_2 norm. Since the energy distance does not depend on the model $g(\cdot; \boldsymbol{\theta})$ nor the loss function $L(\cdot, \cdot)$, a model-independent testing set can be obtained by minimizing the energy distance. The minimizer of the energy distance is referred to as support points

(Mak & Joseph, 2018c):

$$\begin{aligned} \{\mathbf{z}_i^*\}_{i=1}^n &= \arg \min_{\{\mathbf{z}_i\}_{i=1}^n} \overline{\mathbb{E}\mathcal{D}}_{n,N} \\ &= \arg \min_{\{\mathbf{z}_i\}_{i=1}^n} \frac{2}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{z}_i - \mathbf{Z}\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2. \end{aligned} \quad (2.5)$$

Mak and Joseph (2018c) also showed that using support points the error rate of random sampling can be improved by a factor of $\log n$ and can even achieve almost $O(n^{-1})$ convergence in practice.

Thus, support points computed from (2.5) could be used as the testing set, which can work much better than a random sample. This is the idea behind `SPLIT` (Support Points based split). However, there is one issue. The solution to (2.5) need not be a subsample of the original dataset, because the optimization is done on a continuous space. What we really need to do is to solve the following discrete optimization problem:

$$\{\mathbf{U}_i^*\}_{i=1}^n = \arg \min_{\{\mathbf{z}_i\}_{i=1}^n \in \mathcal{D}} \frac{2}{n} \sum_{i=1}^n \mathbb{E} \|\mathbf{z}_i - \mathbf{Z}\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{z}_i - \mathbf{z}_j\|_2. \quad (2.6)$$

Instead of solving (2.6) directly, we proposed to solve it in two steps: first solve the continuous optimization in (2.5) using the difference-of-convex (DC) programming technique (Mak & Joseph, 2018c) to obtain an approximate solution to support points, and then use a sequential nearest neighbor assignment to find the closest points in the dataset to the support points. From here on in, we will refer to this two-step approach as DC-NN, i.e., difference-of-convex programming followed by nearest neighbor assignment. Although much faster than an integer programming solution to (2.6), the computational complexity of this approach is still high.

To arrive at the computational complexity of the DC-NN algorithm, we begin with the DC program that has a complexity of $O(dn(n + N)/P)$ per iteration, where P is the number of

processor cores available. Let $\gamma = n/N$. Then, for τ iterations, we obtain

$$\begin{aligned}
O(\tau dn(n + N)) &= O(\tau d\gamma N(\gamma N + N)/P) \\
&= O(\tau d\gamma(\gamma + 1)N^2/P) \\
&= O(\tau d\gamma N^2/P) \quad \because \gamma \in (0, 1). \tag{2.7}
\end{aligned}$$

The sequential nearest neighbor assignment can be efficiently performed using a `kd-tree`. Building the `kd-tree` is $O(dN \log N)$, and n nearest neighbor queries are needed, each with worst case complexity $O(N^{1-\frac{1}{d}})$ and average case complexity $O(\log N)$. Thus, the average case complexity of the overall procedure can be expressed as

$$\begin{aligned}
O(dN \log N + n \log N + \tau d\gamma N^2/P) &= O(dN \log N + \gamma N \log N + \tau d\gamma N^2/P) \\
&= O((\gamma + d)N \log N + \tau d\gamma N^2/P) \\
&= O(dN \log N + \tau d\gamma N^2/P) \quad \because \gamma < 1. \tag{2.8}
\end{aligned}$$

The quadratic growth in complexity, with respect to the size of the dataset, is the major computational bottleneck of `SPLIT`. Mak and Joseph (2018c) also provide a version of the DC program that uses stochastic majorization-minimization, where the expectations in $\overline{\mathbb{E}\mathcal{D}}_{n,N}$ are computed based on a random sample from \mathcal{D} within each iteration of DC, instead of using the full dataset. Let us denote this procedure as SDC, wherein a random sample of size $\min(\kappa n, N)$ is sampled from \mathcal{D} in every iteration. Thus, for the case of $\gamma < 1/\kappa$, the complexity of SDC for τ iterations reduces to $O(dn(n + \kappa n)/P) = O(\tau d\kappa\gamma^2 N^2/P)$, and the average case complexity of SDC-NN becomes $O(dN \log N + \tau d\kappa\gamma^2 N^2/P)$, which is an improvement over DC-NN for small values of γ . However, for large values of γ , faster algorithms are needed, which leads us to the main topic of this chapter that is discussed in the next section.

2.3 Twinning

The aim of **Twinning** is to partition a dataset into two disjoint sets such that they have similar statistical properties. We will call the two sets as *twins*. The twins needn't be of the same size, but they should have similar statistical distributions. Let $\mathcal{D}^1 = \{\mathbf{U}_i\}_{i=1}^n$ and $\mathcal{D}^2 = \{\mathbf{V}_j\}_{j=1}^{N-n}$ be the twins such that $\mathcal{D}^1 \cap \mathcal{D}^2 = \emptyset$ and $\mathcal{D}^1 \cup \mathcal{D}^2 = \mathcal{D}$. Let $\overline{\mathbb{E}\mathbb{D}}_{n,N-n}$ be the energy distance between \mathcal{D}^1 and \mathcal{D}^2 , which is given by

$$\overline{\mathbb{E}\mathbb{D}}_{n,N-n} := \frac{2}{n(N-n)} \sum_{i=1}^n \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 - \frac{1}{(N-n)^2} \sum_{i=1}^{N-n} \sum_{j=1}^{N-n} \|\mathbf{V}_i - \mathbf{V}_j\|_2.$$

The twins are obtained by minimizing $\overline{\mathbb{E}\mathbb{D}}_{n,N-n}$ with respect to \mathcal{D}^1 and \mathcal{D}^2 , i.e.,

$$\begin{aligned} \{\mathbf{U}_i^*\}_{i=1}^n, \{\mathbf{V}_j^*\}_{j=1}^{N-n} &= \arg \min_{\{\mathbf{U}_i\}_{i=1}^n, \{\mathbf{V}_j\}_{j=1}^{N-n}} \overline{\mathbb{E}\mathbb{D}}_{n,N-n} \\ &\text{subject to: } \{\mathbf{U}_i\}_{i=1}^n \cap \{\mathbf{V}_j\}_{j=1}^{N-n} = \emptyset \\ &\quad \{\mathbf{U}_i\}_{i=1}^n \cup \{\mathbf{V}_j\}_{j=1}^{N-n} = \mathcal{D}. \end{aligned} \tag{2.9}$$

At first sight, **Twinning** appears to be a much harder problem than **SPLIT** since we need to perform the optimization in Nd variables instead of nd variables. Interestingly, the following result shows that the objectives of **Twinning** and **SPLIT** are equivalent.

Proposition 1. *Given $n = \gamma N$, $\overline{\mathbb{E}\mathbb{D}}_{n,N} = (1 - \gamma)^2 \cdot \overline{\mathbb{E}\mathbb{D}}_{n,N-n}$*

All the proofs are given in the Appendix. Proposition 1 shows that minimizing the energy distance between \mathcal{D}^1 and \mathcal{D}^2 is the same as minimizing the energy distance between \mathcal{D}^1 and \mathcal{D} . Thus, to solve (2.9), we can solve (2.6) and set $\{\mathbf{V}_j^*\}_{j=1}^{N-n} = \mathcal{D} \setminus \{\mathbf{U}_i^*\}_{i=1}^n$. However, as formally stated below, this is a difficult optimization problem.

Proposition 2. *The optimization in (2.6) is \mathcal{NP} -hard.*

Hence, from Propositions 1 and 2, we have that solving (2.9) to optimality is intractable. An efficient algorithm to obtain a reasonable solution to (2.9) is presented next.

2.3.1 Algorithm

There are three parts to $\overline{\text{ED}}_{n,N-n}$ that we will examine individually, they are

$$\begin{aligned}
\overline{\text{ED}}_{n,N-n}^1 &:= \frac{2}{n(N-n)} \sum_{i=1}^n \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2, \\
\overline{\text{ED}}_{n,N-n}^2 &:= -\frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2, \text{ and} \\
\overline{\text{ED}}_{n,N-n}^3 &:= -\frac{1}{(N-n)^2} \sum_{i=1}^{N-n} \sum_{j=1}^{N-n} \|\mathbf{V}_i - \mathbf{V}_j\|_2.
\end{aligned} \tag{2.10}$$

Minimizing $\overline{\text{ED}}_{n,N-n}^1$ translates to bringing \mathcal{D}^1 and \mathcal{D}^2 closer to each other, whereas minimizing $\overline{\text{ED}}_{n,N-n}^2$ and $\overline{\text{ED}}_{n,N-n}^3$ causes the points in \mathcal{D}^1 and \mathcal{D}^2 , respectively, to be spread out. Consider the case of CADEX and SPXY that were alluded to in Section 2.1, both procedures effectively aim at minimizing $\overline{\text{ED}}_{n,N-n}^3$ alone, i.e., a well spread out \mathcal{D}^2 is produced, while nothing can be said about \mathcal{D}^1 and its statistical similarity to \mathcal{D}^2 . DUPLEX on the other hand focuses on minimizing both $\overline{\text{ED}}_{n,N-n}^2$ and $\overline{\text{ED}}_{n,N-n}^3$, still neglecting $\overline{\text{ED}}_{n,N-n}^1$. DUPLEX is an improvement over CADEX because in essence \mathcal{D}^1 is now a relatively rigorous validation set as it is more spread out and potentially includes extreme points. When designing the Twinning algorithm, we will incorporate $\overline{\text{ED}}_{n,N-n}^1$ as well, thereby tying all the three pieces together.

Assume that $\gamma = 1/r$, where r is an integer, i.e., $N = rn$. The dataset \mathcal{D} can now be partitioned into n disjoint subsets each with r points in it. At this stage, if a single point is selected from each of the n subsets into \mathcal{D}^1 and the remaining $r - 1$ points into \mathcal{D}^2 , the desired splitting ratio γ is achieved. What remains now is to perform the said partitioning, and the selection within the n resulting subsets in a manner that respects the $\overline{\text{ED}}_{n,N-n}$ criterion. With this partitioning in mind, we propose a sequential approach where given a starting position $\mathbf{u}_1 \in \mathcal{D}$, the $r - 1$ closest points to \mathbf{u}_1 in \mathcal{D} together with \mathbf{u}_1 form the first of the n subsets. Let this first subset be $\mathcal{S}_1 = \{\mathbf{u}_1, \mathbf{v}_1^1, \dots, \mathbf{v}_1^{r-1}\} \in \mathcal{D}$, here \mathbf{u}_1 is pushed

into \mathcal{D}^1 and the remaining $\mathbf{v}_1^1, \dots, \mathbf{v}_1^{r-1}$ are pushed into \mathcal{D}^2 . For the sake of demonstration, consider a simple two-dimensional dataset with $N = 50$ observations generated as follows: $X_i \stackrel{iid}{\sim} N(0, 1)$ and $Y_i|X_i \stackrel{iid}{\sim} N(X_i^2, 1)$ for $i = 1, \dots, N$. We have that $r = 5$ for an 80-20 split. Plot (a) in Figure 2.2 depicts the first subset \mathcal{S}_1 , given that we start from the encircled point labelled as ‘1’ which represents \mathbf{u}_1 .

Next, we select $\mathbf{u}_2 \in \mathcal{D} \setminus \mathcal{S}_1$ based on some selection rule, and then proceed to identify the $r - 1$ closest points to \mathbf{u}_2 in $\mathcal{D} \setminus \mathcal{S}_1$. Let $\mathcal{S}_1 = \{\mathbf{u}_1, \mathbf{v}_1^1, \dots, \mathbf{v}_1^{r-1}\}$ be such that $\|\mathbf{u}_1 - \mathbf{v}_1^{k-1}\|_2 \leq \|\mathbf{u}_1 - \mathbf{v}_1^k\|_2, \forall k = 2, \dots, r - 1$, i.e., \mathbf{v}_1^1 is the closest neighbor to \mathbf{u}_1 in \mathcal{S}_1 , while \mathbf{v}_1^{r-1} is the farthest. Here we propose a selection rule where we select the closest point to \mathbf{v}_1^{r-1} in $\mathcal{D} \setminus \mathcal{S}_1$ as \mathbf{u}_2 . Similar to the first subset, let the second subset be $\mathcal{S}_2 = \{\mathbf{u}_2, \mathbf{v}_2^1, \dots, \mathbf{v}_2^{r-1}\} \in \mathcal{D} \setminus \mathcal{S}_1$ where \mathbf{u}_2 is pushed into \mathcal{D}^1 and the remaining $\mathbf{v}_2^1, \dots, \mathbf{v}_2^{r-1}$ are pushed into \mathcal{D}^2 . Plot (b) in Figure 2.2 shows both subsets \mathcal{S}_1 and \mathcal{S}_2 where the encircled point labelled as ‘2’ is \mathbf{u}_2 . Now we see the pattern where given \mathcal{S}_{i-1} , we select the closest point to \mathbf{v}_{i-1}^{r-1} in $\mathcal{D} \setminus \cup_{j=1}^{i-1} \mathcal{S}_j$ to be \mathbf{u}_i . It can happen that when γ is restricted such that $1/\gamma$ is an integer, $n \neq \gamma N$. In such scenarios, it is straightforward to set $n \leftarrow \lceil \gamma N \rceil$, which results in the cardinality of the final subset to be strictly lower than $r = 1/\gamma$, i.e., $|\mathcal{S}_n| < r$. Algorithm 2 formally states the **Twinning** algorithm. Since the sample dataset has 50 points, for an 80-20 split, we need to identify 10 subsets: $\mathcal{S}_1, \dots, \mathcal{S}_{10}$ each with 5 points in it as described. Figure 2.2 depicts the subsets identified at the end of iterations 1, 2, 5, and 10 of **Twinning**. In plot (d) of Figure 2.2, the encircled points make up \mathcal{D}^1 , and the remaining points form \mathcal{D}^2 .

Twinning attempts to minimize all three parts of $\overline{\mathbb{E}\mathbb{D}}_{n, N-n}$ that are outlined in (2.10). It is immediately seen that, when the r points in each of the n subsets are closer to each other, we end up reducing $\overline{\mathbb{E}\mathbb{D}}_{n, N-n}^1$ since for most testing points there are at least $r - 1$ of its neighbors in \mathcal{D}^2 . In other words, for most testing points there will not be another testing point closer than its $(r - 1)^{th}$ neighbor, which in turn brings down $\overline{\mathbb{E}\mathbb{D}}_{n, N-n}^2$. Reduction in $\overline{\mathbb{E}\mathbb{D}}_{n, N-n}^3$ is given if \mathcal{D}^2 is well spread out rather than aggregated in a region. With the proposed selection rule, **Twinning** sequentially finds subsets $\mathcal{S}_i, \forall i = 1, \dots, n$ such that most subsets are expected

to be adjacent to their previous subset with minimal overlaps. This in turn ensures that \mathcal{D}^2 , which constitutes all but one point from each subset, sufficiently covers the region of the dataset, thereby reducing $\overline{\text{ED}}_{n,N-n}^3$ as alluded.

Algorithm 2 Twinning [R package: `twinning`]

```

1: Input  $\mathcal{D} \in \mathbb{R}^{N \times d}$ , splitting ratio  $\gamma : 1/\gamma \in \mathbb{Z}$ , and starting position  $\mathbf{u}_1$ 
2: Standardize the columns of  $\mathcal{D}$ 
3:  $n \leftarrow \lceil \gamma N \rceil$ ;  $r \leftarrow 1/\gamma$ ;  $\mathcal{D}^1 \leftarrow \{\}$ ;  $\mathcal{D}^2 \leftarrow \{\}$ 
4: for  $i \in \{1, \dots, n\}$  do
5:   if  $i \neq 1$  then
6:      $\mathbf{u}_i = \arg \min_{\mathbf{u} \in \mathcal{D}} \{\|\mathbf{u} - \mathbf{v}_{i-1}^{r-1}\|_2\}$ 
7:   end if
8:   if  $i = n$  then
9:      $r \leftarrow \lfloor \mathcal{D} \rfloor$ 
10:  end if
11:   $\{\mathbf{v}_i^1, \dots, \mathbf{v}_i^{r-1}\} = \arg \min_{\{\mathbf{v}^1, \dots, \mathbf{v}^{r-1}\} \in \mathcal{D} \setminus \{\mathbf{u}_i\}} \{\sum_{j=1}^{r-1} \|\mathbf{v}^j - \mathbf{u}_i\|_2\}$ 
12:   $\mathcal{D}^1 \leftarrow \mathcal{D}^1 \cup \{\mathbf{u}_i\}$ ;  $\mathcal{D}^2 \leftarrow \mathcal{D}^2 \cup \{\mathbf{v}_i^1, \dots, \mathbf{v}_i^{r-1}\}$ ;  $\mathcal{D} \leftarrow \mathcal{D} \setminus \{\mathbf{u}_i, \mathbf{v}_i^1, \dots, \mathbf{v}_i^{r-1}\}$ 
13: end for
14: return  $\mathcal{D}^1, \mathcal{D}^2$ 

```

Figure 2.3 shows a random 80-20 split, a split from 500 iterations of DC-NN, as well as the split obtained by Twinning on the same sample dataset. When comparing different splits of a fixed dataset, it is easier to use $\overline{\text{ED}}_{n,N}$ as the metric, i.e., energy distance between the smaller twin \mathcal{D}^1 and the whole dataset \mathcal{D} . Similar to how $\overline{\text{ED}}_{n,N-n}$ in (2.10) has three parts, $\overline{\text{ED}}_{n,N}$ also has three parts, of which the third remains constant if the dataset is fixed. Hence, for the remainder of this chapter, the energy that is being referred to in the plots is

$$\overline{\text{ED}}_{n,N}^{1,2} := \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{U}_i - \mathbf{Z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2, \quad (2.11)$$

where $\{\mathbf{Z}_i\}_{i=1}^N$ is the dataset and $\{\mathbf{U}_i\}_{i=1}^n$ is the testing set as before. $\overline{\text{ED}}_{n,N}^{1,2}$ for the split obtained from Twinning in Figure 2.2 is 1.724636. We see that the performance of Twinning comes close to that of DC-NN. In Section 2.4, we provide an extensive comparison between Twinning and DC-NN, where we find that in higher dimensions and for larger values of N , Twinning edges out DC-NN with substantially lower computation time.

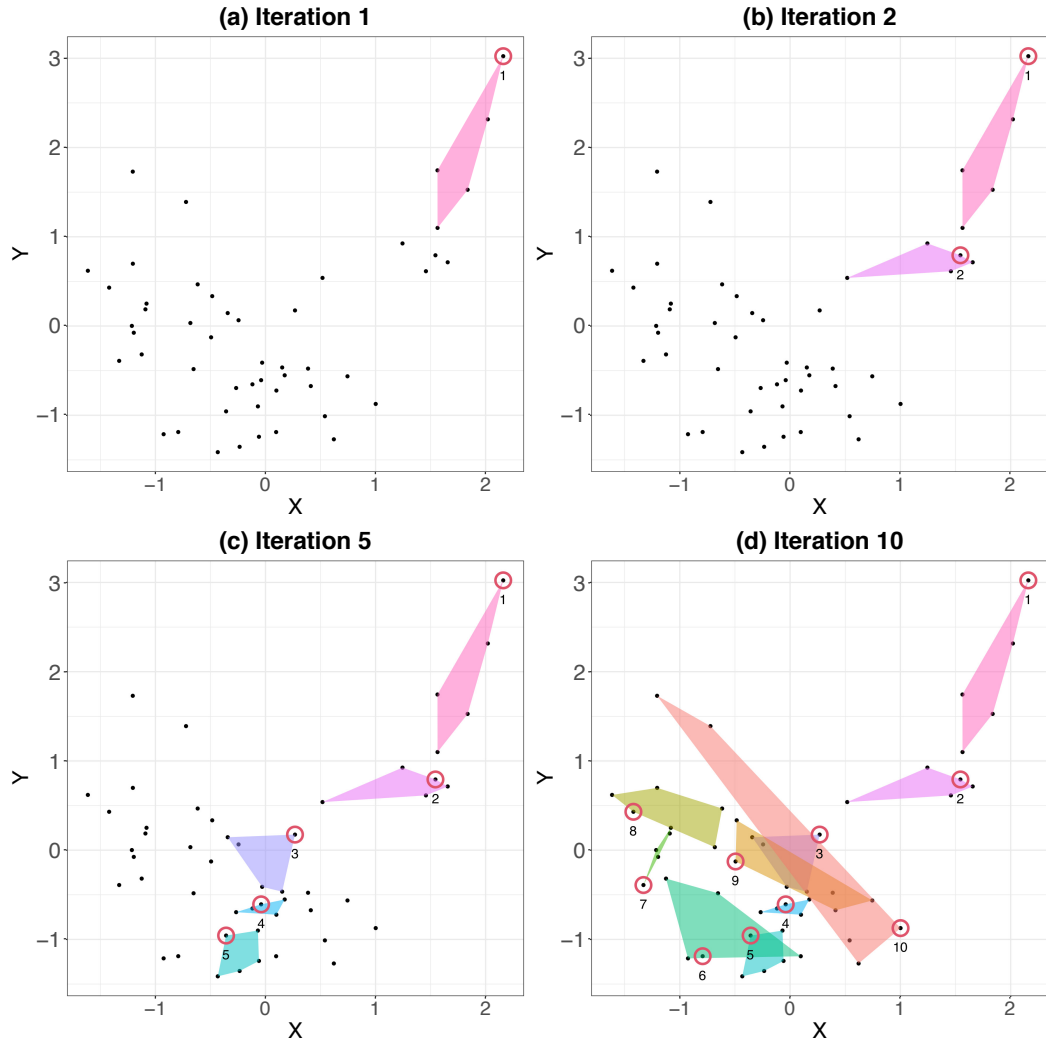


Figure 2.2: The convex hull of subsets identified by **Twinning** at the end of iterations 1, 2, 5, and 10 for the sample dataset described in Section 2.3. Points in \mathcal{D}^1 are shown as encircled points, and they are numbered in the order they were selected.

2.3.2 Computational Complexity

Much of **Twinning**'s computational complexity can be attributed to nearest neighbor queries. Exact nearest neighbor queries in higher dimensions may not be efficient, and several approximate nearest neighbor search algorithms have been proposed in the literature to address this, e.g., locality-sensitive hashing (Slaney & Casey, 2008). Li et al. (2019) provides a comprehensive evaluation of nineteen such algorithms. Nevertheless, following the implementation of **DC-NN** (Vakayil et al., 2021) that uses a **kd-tree** based exact nearest

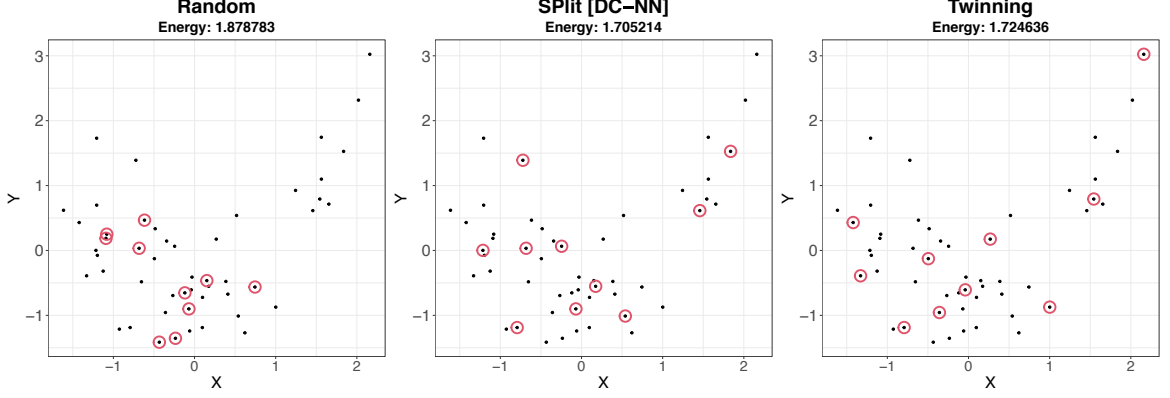


Figure 2.3: A random split of the dataset described in Section 2.3 is shown on the left, while a SPlit obtained using DC-NN is shown in the middle, and with Twinning on the right. The encircled points constitute \mathcal{D}^1 in each of the three plots.

neighbor search (Blanco & Rai, 2014), we use the same in our analysis of Twinning.

Building the kd-tree is $O(dN \log N)$. In each of the n iterations, Twinning makes one nearest neighbor query to locate \mathbf{u}_i , $i \neq 1$, and one r -nearest neighbors query to identify $\{\mathbf{v}_i^1, \dots, \mathbf{v}_i^{r-1}\}$; the former has a worst case complexity of $O(N^{1-\frac{1}{d}})$, and $O(N^{1-\frac{1}{d}} + r)$ for the latter. The complexity of these nearest neighbor(s) queries is lower in practice, with the average case complexity being $O(\log N)$ (J. H. Friedman et al., 1977). Updating the kd-tree after each query is an expensive affair, hence, instead of deleting a point after it has been queried out of the tree, the point is merely masked. The overall complexity of Twinning in the worst case can be simplified to

$$\begin{aligned}
 O(dN \log N + 2n(N^{1-\frac{1}{d}} + r)) &= O(dN \log N + 2\gamma N(N^{1-\frac{1}{d}} + \frac{1}{\gamma})) \\
 &= O(dN \log N + 2(\gamma N^{2-\frac{1}{d}} + N)) \\
 &= O(dN \log N + \gamma N^{2-\frac{1}{d}}),
 \end{aligned}$$

while the average case complexity is

$$O(dN \log N + 2n \log N) = O(dN \log N + 2\gamma N \log N)$$

$$\begin{aligned}
&= O((\gamma + d)N \log N) \\
&= O(dN \log N) \quad \because \gamma < 1,
\end{aligned} \tag{2.12}$$

which is much better than the complexity of DC-NN given in (2.8).

2.4 Experiments

Consider d -dimensional datasets with N rows generated by sampling from a multivariate normal with $\boldsymbol{\mu} = [0, \dots, 0]^\top \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d} : \Sigma_{ij} = 0.5^{|i-j|}, \forall i, j \in \{1, \dots, d\}$. In this section we compare DC-NN and **Twinning** for splitting these datasets. The performance of the splits is measured in terms of the computation time and the energy distance metric $\overline{\text{ED}}_{n,N}^{1,2}$ defined in (2.11). Plot (a) in Figures 2.4, 2.5, and 2.6 considers smaller datasets, where the experiments are run on a laptop with 6-core Intel 2.6 GHz processor, while plot (b) considers bigger datasets, where the experiments are run on a 36-core Intel 3.0 GHz processor. The plots show the ratio of $\overline{\text{ED}}_{n,N}^{1,2}$ obtained from **Twinning** to that obtained from DC-NN; hence, an energy ratio of less than 1 for a given N , d , and γ indicates that **Twinning** performed better than DC-NN with respect to the quality of the split.

Twinning is deterministic when \mathbf{u}_1 is fixed, and in this section \mathbf{u}_1 is chosen to be the point farthest from the centroid of the dataset. With DC-NN, the split can vary over multiple runs of the algorithm owing to its random initialization, hence, for stability, 500 iterations of the iterative algorithm that computes support points is used; the quality of the split improves with every iteration of the iterative algorithm. Moreover, DC-NN can be run in parallel, with the computation time being inversely proportional to the number of parallel threads, whereas **Twinning** is serial. Still, we see that the computation times for **Twinning** are significantly lower compared to DC-NN. For example, consider plot (b) of Figure 2.4, where for $N = 10^5$, $d = 16$, and $\gamma = 0.2$, the computation time for **Twinning** is better than DC-NN by a factor of 27, even though DC-NN is executed in parallel on 36 cores, i.e., if DC-NN were to be executed on a single core, **Twinning** would observe an improvement in computation time by a factor

of 27×36 over DC-NN. In addition, the quality of the split obtained by **Twinning** is better than that obtained from DC-NN.

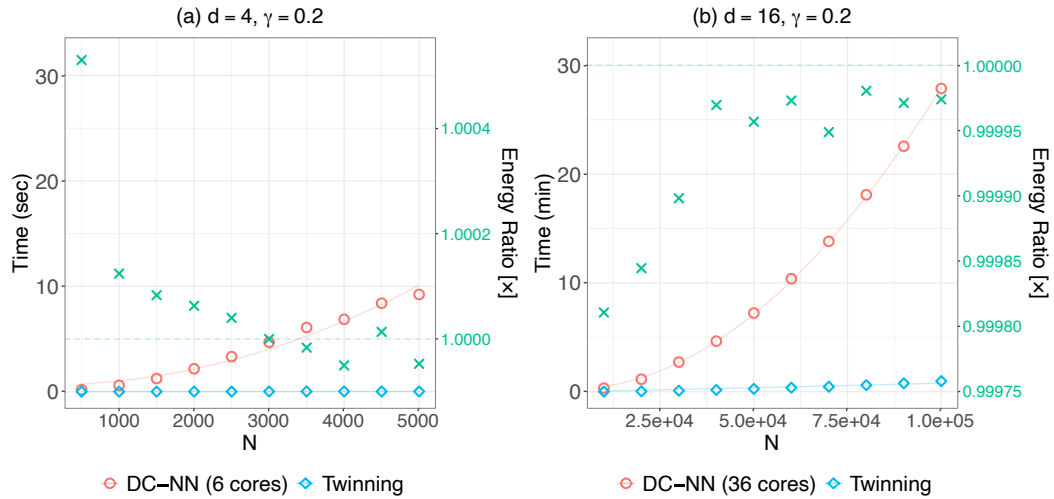


Figure 2.4: Comparison between DC-NN and Twinning for varying N with fixed d and γ . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and Twinning, respectively. The green crosses denote the energy ratio of Twinning to DC-NN.

From Figures 2.4, 2.5, and 2.6, it is evident that **Twinning** outperforms DC-NN, both in terms of computation time and quality of the split, when it comes to bigger datasets. It is when N , d , and γ are all small, DC-NN produces better quality splits than **Twinning**. Hence, for partitioning Big Data, **Twinning** is the algorithm of choice.

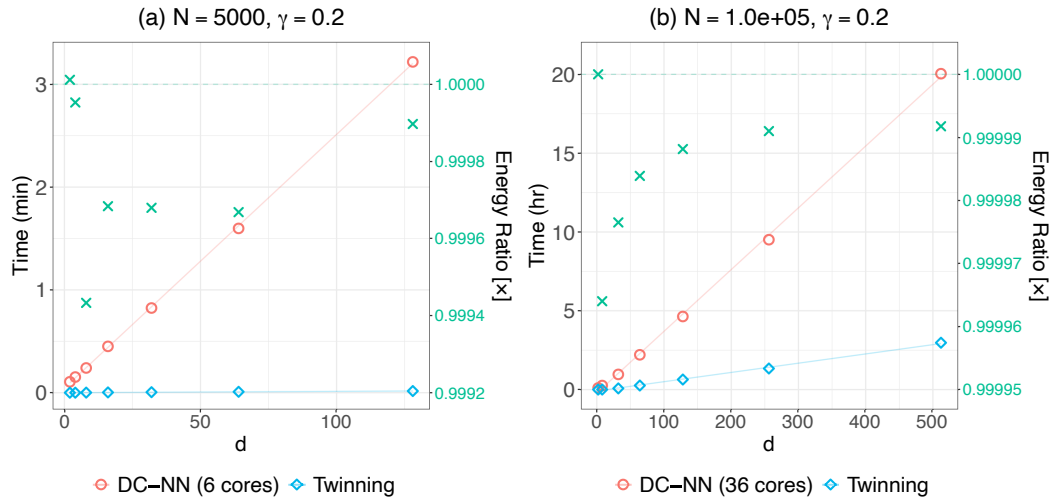


Figure 2.5: Comparison between DC-NN and Twinning for varying d with fixed N and γ . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and Twinning, respectively. The green crosses denote the energy ratio of Twinning to DC-NN.

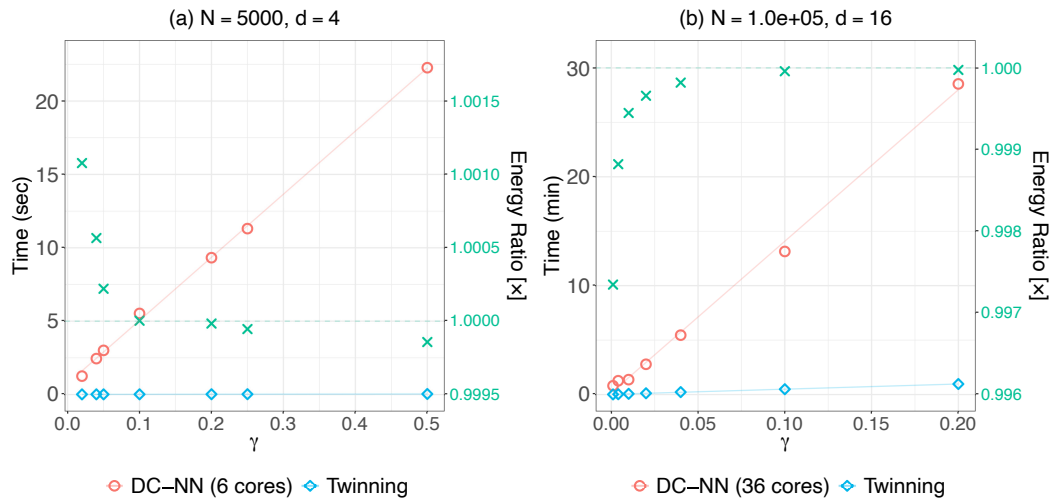


Figure 2.6: Comparison between DC-NN and Twinning for varying γ with fixed N and d . 500 iterations are used for DC-NN. The red circles and blue diamonds represent the computation times for DC-NN and Twinning, respectively. The green crosses denote the energy ratio of Twinning to DC-NN.

2.5 Multiplets

It is often desired to split a dataset into multiple disjoint sets. In keeping with the terminology thus far, if we divide the dataset into three sets with similar statistical properties, we call them *triplets*, four sets as *quadruplets*, and so on. In general, we will call the sets *multiplets*.

There are several applications for multiplets. The well studied k -fold cross validation approach proceeds by randomly splitting a dataset into k sets (Hastie et al., 2009). Moreover, with the technological advances in parallel computing, divide-and-conquer methods that act upon separate blocks of a dataset are becoming increasingly popular for statistical analysis on Big Data (Guha et al., 2012).

When it comes to k -fold cross validation, an estimate of the generalization error of a given model is made based on the individual error estimates on the k sets (Rodriguez et al., 2010). The reliability of these k separate error estimates are bound to improve when the k sets are themselves distributed similar to the whole dataset. The divide-and-conquer methods split Big Data into, say, k manageable sets that are analyzed separately and the results are carefully merged to form a combined inference on the Big Data, thereby circumventing the storage and computational limits of analyzing Big Data on a single machine. It is only reasonable to assume that when these k separate sets are distributed similar to their union, i.e., the Big Data, the quality of the overall inference could improve, albeit further research is needed in this regard.

Here we demonstrate how **Twinning** can be adapted to split a given dataset $\mathcal{D} \in \mathbb{R}^{N \times d}$ into k multiplets: $\mathcal{D}^1, \mathcal{D}^2, \dots, \mathcal{D}^k$ such that $\cup_{i=1}^k \mathcal{D}^i = \mathcal{D}$ and $\mathcal{D}^i \cap \mathcal{D}^j = \emptyset, \forall i \neq j$. For ease of presentation, we will explain the methodology to generate multiplets of the same cardinality, i.e., $|\mathcal{D}^i| = N/k, \forall i$. As defined in (2.11), let $\overline{\mathbb{E}\mathcal{D}}_{n_i, N}^{1,2}$ be the energy of \mathcal{D}^i with respect to $\mathcal{D}, \forall i$, and let $\overline{\mathbb{E}\mathcal{D}}_{n, N}^{1,2(*)} := \max\{\overline{\mathbb{E}\mathcal{D}}_{n_i, N}^{1,2} : i \in \{1, \dots, k\}\}$. A lower $\overline{\mathbb{E}\mathcal{D}}_{n, N}^{1,2(*)}$ indicates that all the k sets $\mathcal{D}^i, \forall i$ are distributed similar to \mathcal{D} . We propose the following three strategies to generate multiplets with **Twinning**,

S¹ Run **Twinning** on \mathcal{D} with $\gamma = \frac{n}{N} = \frac{1}{k}$ to obtain \mathcal{D}^1 and $\mathcal{D} \setminus \mathcal{D}^1$. Next, run **Twinning** on $\mathcal{D} \setminus \mathcal{D}^1$ with $\gamma = 1/(k-1)$ to obtain \mathcal{D}^2 and $\mathcal{D} \setminus \cup_{i=1}^2 \mathcal{D}^i$. Repeat until \mathcal{D}^{k-1} and \mathcal{D}^k are obtained.

S² Repeatedly run **Twinning** on \mathcal{D} with $\gamma = 0.5$ until all the k sets $\mathcal{D}^1, \dots, \mathcal{D}^k$ are obtained, assuming k is a power of 2, i.e., $k = 2^a : a \in \mathbb{Z}$.

S³ Run **Twinning** on \mathcal{D} with $\gamma = \frac{1}{k}$. Let $\mathcal{S}_e, \forall e \in \{1, \dots, n\}$ be the n mutually exclusive and collectively exhaustive subsets of \mathcal{D} identified by **Twinning**. Let $\mathcal{S}_e = \{\mathbf{u}_e, \mathbf{v}_e^1, \dots, \mathbf{v}_e^{r-1}\}$, where $r = k$. For subset $\mathcal{S}_e, \forall e \in \{1, \dots, n\}$, add \mathbf{u}_e to $\mathcal{D}^1, \mathbf{v}_e^1$ to \mathcal{D}^2, \dots , and \mathbf{v}_e^{r-1} to \mathcal{D}^k .

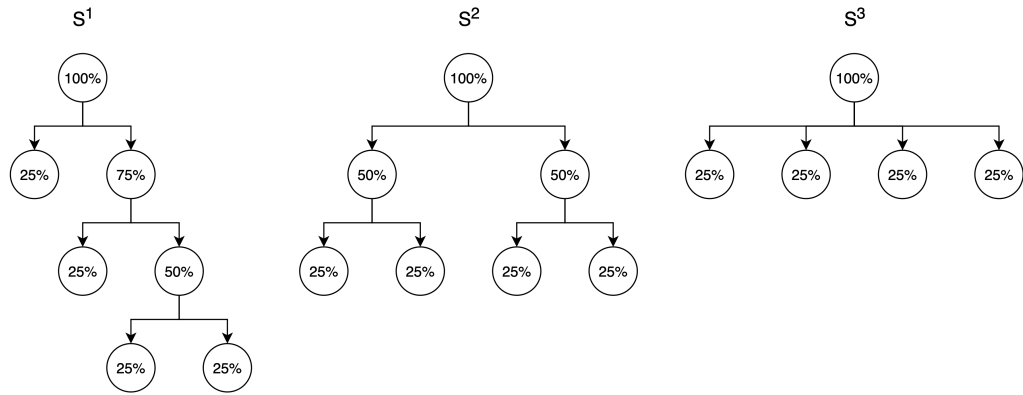


Figure 2.7: Pictorial representation of the three proposed strategies to generate quadruplets of the same cardinality with **Twinning**.

Figure 2.7 provides a simple depiction of the three strategies **S¹**, **S²**, and **S³** for generating quadruplets, i.e., $k = 4$. We see that both **S¹** and **S²** require $k - 1$ runs of **Twinning**, while strategy **S³** gets away with a single run of **Twinning** on \mathcal{D} . To assess the performance of the three strategies, consider a $d = 16$ dimensional dataset with $N = 10,000$ rows generated by sampling from a multivariate normal with $\boldsymbol{\mu} = [0, \dots, 0]^T \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d} : \Sigma_{ij} = 0.5^{|i-j|}, \forall i, j \in \{1, \dots, d\}$. We generate quadruplets of this dataset such that $n_i = 2500, \forall i \in \{1, 2, 3, 4\}$ using **S¹**, **S²**, and **S³**, wherein **Twinning** starts from a random \mathbf{u}_1

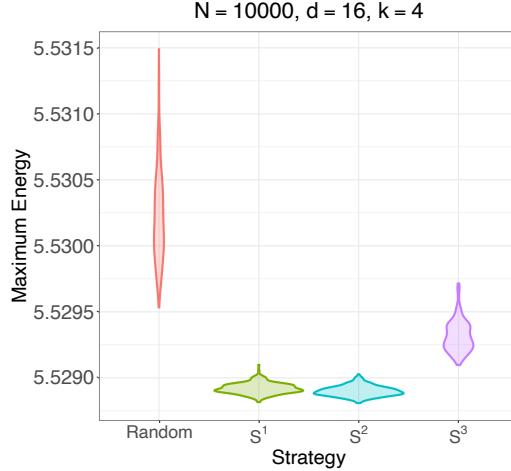


Figure 2.8: Distribution of $\overline{\mathbb{E}D}_{n,N}^{1,2(*)}$ over 250 multi-splits of the multivariate normal dataset by random splitting, and the three proposed strategies for generating multiplets with **Twinning**.

in every run. We also make a multi-split randomly for comparison. This experiment is then repeated 250 times on the same dataset, and Figure 2.8 reports the distribution $\overline{\mathbb{E}D}_{n,N}^{1,2(*)}$ over these 250 multi-splits with the four strategies that includes random splitting. It is readily observed that all the three strategies \mathbf{S}^1 , \mathbf{S}^2 , and \mathbf{S}^3 perform better than randomly splitting the dataset into 4 sets such that $\overline{\mathbb{E}D}_{n,N}^{1,2(*)}$ is minimized. In addition, we see that strategies \mathbf{S}^1 and \mathbf{S}^2 perform better than \mathbf{S}^3 , at the computational cost of additional **Twinning** runs. However, \mathbf{S}^3 can only produce multiplets of same cardinality, whereas it is straightforward to generalize \mathbf{S}^1 and \mathbf{S}^2 for the case of unequal cardinalities by varying the γ in the **Twinning** runs.

2.6 Applications

In this section, we discuss several applications of **Twinning**.

2.6.1 Data Splitting

Consider the problem of predicting taxicab trip duration from the trip characteristics such as total trip distance and proportion of highway. Here, we use the New York City (NYC)

taxicab dataset provided as part of the 2017 kaggle competition. In particular, we use the dataset from Benmeida (2017) that contains 2,074,291 observations. We will use eight continuous features for modeling, as done in Joseph and Mak (2021).

Before fitting the model, we will make an 80-20 split of the dataset, i.e., training set will contain 1,659,432 observations and testing set 414,859 observations. If we were to use DC-NN, splitting alone would have taken about one month to finish on our laptop. SDC-NN would also take similar amount of time because it is the same as DC-NN when $\kappa \geq 5$ and $\gamma = 0.2$. Thus, it is impractical to optimally split this big dataset using DC-NN or SDC-NN. On the other hand, **Twinning** was able to make the split in just about two minutes! It is clear that without **Twinning** one would have to be content with random subsampling. Thus, the remarkable speed of **Twinning** makes optimal data splitting a reality, especially for Big Data problems.

2.6.2 Data Compression

In the current data-rich age, computational resources are a bottleneck to analyze or store the vast amount data that is being generated across domains. Furthermore, the trend is set to continue into the foreseeable future, given the lackluster growth rate of computational power over the decade, as we await technological breakthroughs (Theis & Wong, 2017). Hence, we resort to data compression methodologies that attempt to reduce the size of Big Data while retaining complete or partial information from the Big Data. There exists a fundamental limit to lossless data compression (Shannon, 1948), i.e., there is a limit to the extent which a given Big Data can be reduced in size such that no information is lost in the process. The trade-off between the reduction in size and the amount of information retained is what characterizes a lossy data compression that allows for further reduction in size at the expense of information loss.

Twinning can be viewed as a lossy data compression methodology, where a statistically similar subsample is obtained from the Big Data after compression. The statistical similarity

can be associated with retention of information, i.e., the more statistically similar a given subsample is to the Big Data, the more information is retained. Either of the twin subsamples produced by `Twinning` can be the subsample in question, that can be used instead of the Big Data for tractable statistical analysis. As we will see, the computational efficiency of `Twinning` is a major factor that enables its use for data compression.

Since `Twinning` is based on support points, it can serve as a model-independent data compression methodology, as opposed to the many model-based compression methods in the literature, e.g., information-based optimal subsample (IBOSS) by Wang et al. (2019). Additionally, unlike support points, `Twinning` produces a subsample of the original dataset, and thus, it could be viewed as a data reduction technique in addition to data compression. We note that, although `Twinning` makes use of the response column(s) in the dataset, it still behaves like an unsupervised data reduction technique, which is quite different from supervised methods for data reduction, e.g., `supercompress` by Joseph and Mak (2021).

To demonstrate the applicability of `Twinning` for data compression, consider again the NYC taxi trip dataset introduced in Section 2.6.1. The training and testing sets have 1,659,432 and 414,859 observations, respectively. Suppose we are interested in fitting a random forest regression model using the `randomForest` package in R (Liaw & Wiener, 2002). Fitting a random forest under the default settings of this package runs into memory allocation issues. Therefore, data compression is a must to train the random forest. Since it is outright impractical to employ DC-NN for the same task, we will compare the performance of `Twinning` with random subsampling and SDC-NN with $\kappa = 10$. Consider 90% compression, wherein we apply `Twinning` on the training set with $\gamma = 0.1$. On a laptop with 6-core Intel 2.6 GHz processor, `Twinning` takes about 40 seconds to reduce the training set, and it takes 24 minutes to train the random forest on the reduced training set that has 165,944 observations. However, SDC-NN would have taken more than 11 days for the same task and therefore, it is impractical to use SDC-NN for data compression.

Table 2.1 lists the total modeling time under the three compression methods: random

γ	Reduction Time			Training Time	Total Modeling Time		
	Random	SDC-NN	Twinning		Random	SDC-NN	Twinning
0.1	0	11* days	40 sec	24 min	24 min	11* days	\approx 24 min
0.05	0	$2\frac{1}{2}$ * days	28 sec	7 min	7 min	$2\frac{1}{2}$ * days	\approx 7 min
0.01	0	153 min	15 sec	27 sec	27 sec	\approx 153 min	42 sec
0.005	0	38 min	14 sec	11 sec	11 sec	\approx 38 min	25 sec
0.001	0	101 sec	13 sec	2 sec	2 sec	103 sec	15 sec

Table 2.1: Time taken to reduce the NYC taxi trip training set, and then fitting a random forest regression, on a laptop with 6-core Intel 2.6 GHz processor. Fields marked with an * are estimates.

subsampling, SDC-NN ($\kappa = 10$), and Twinning, for γ values: 0.10, 0.05, 0.01, 0.005, and 0.001. We can see that the performance of Twinning is quite remarkable—it is almost as fast as the random subsampling-based modeling. On the other hand, SDC-NN is clearly not a feasible data compression method when γ is large. To compare the testing performance of the fitted models under the three methods, we repeated the experiment 50 times for $\gamma \in \{0.005, 0.001\}$. The testing performance is measured in terms of the root-mean-squared prediction error (RMSE) on the testing set that has 414,859 observations. Figure 2.9 provides the distribution of RMSE over the 50 repetitions, where we see that the testing performance of the random forest models fitted with Twinning edges out those with random subsampling and SDC-NN. It is quite amazing to see that Twinning outperforms SDC-NN not only in computational time, but also in the quality of splits.

2.6.3 Cross Validation

In this section, we demonstrate how multiplets can be applied to cross validation, as alluded to in Section 2.5. Consider the airfoil self-noise dataset (Brooks et al., 1989) from the UCI Machine Learning Repository. The dataset was originally developed at NASA after a series of aerodynamic and acoustic experiments on airfoil blade sections. The dataset has 1,503 observations with five continuous features and a continuous response indicating

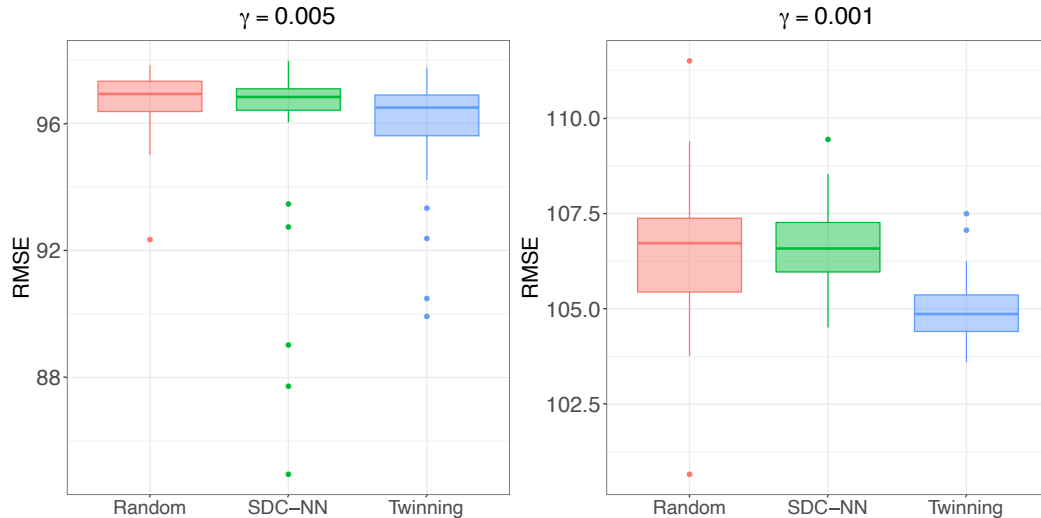


Figure 2.9: Distribution of testing RMSE over 50 repetitions of compressing the NYC taxi trip training set with random subsampling, SDC-NN ($\kappa = 10$), and Twinning, followed by fitting a random forest regression model.

the sound pressure level in decibels. We will fit a regression model on the dataset with LASSO (Tibshirani, 1996), where the regularization parameter λ is estimated by k -fold cross validation.

Conventionally, the k folds are obtained by randomly partitioning the dataset into k sets of similar size. Here, we analyze the effect of using multiplets, instead of the random partitions as the k folds, e.g., for 4-fold and 8-fold cross validation, we can use quadruplets and octuplets, respectively. The `glmnet` (J. Friedman et al., 2010) package in R is used to fit LASSO, where the same sequence of λ values is used when comparing the performance with random folds and multiplets. Let λ_{min} be the λ corresponding to minimum mean-squared cross validation error (MSCV). Depending upon the k folds supplied to LASSO, the estimated value of λ_{min} can vary. Consider fitting LASSO on the given dataset with 4-fold and 8-fold cross validation, the left panel of Figure 2.10 depicts the distribution λ_{min} estimated by LASSO, over 500 distinct random folds and multiplets, while the right panel shows the distribution of MSCV at λ_{min} . The multiplets used here are generated by Twinning using the S^2 strategy.

It is readily observed from Figure 2.10 that the use of multiplets for cross validation leads to a stable estimate for the regularization parameter, alongside consistent and smaller MSCV, when compared to random folds. Another interesting takeaway is that even the 4-fold cross validation with multiplets performs better than 8-fold cross validation using random folds, which could prove to be a major computational advantage for tuning computationally expensive models using cross-validation. However, more theoretical investigation and computational experiments are needed to understand this better, which we leave as a topic for future research.

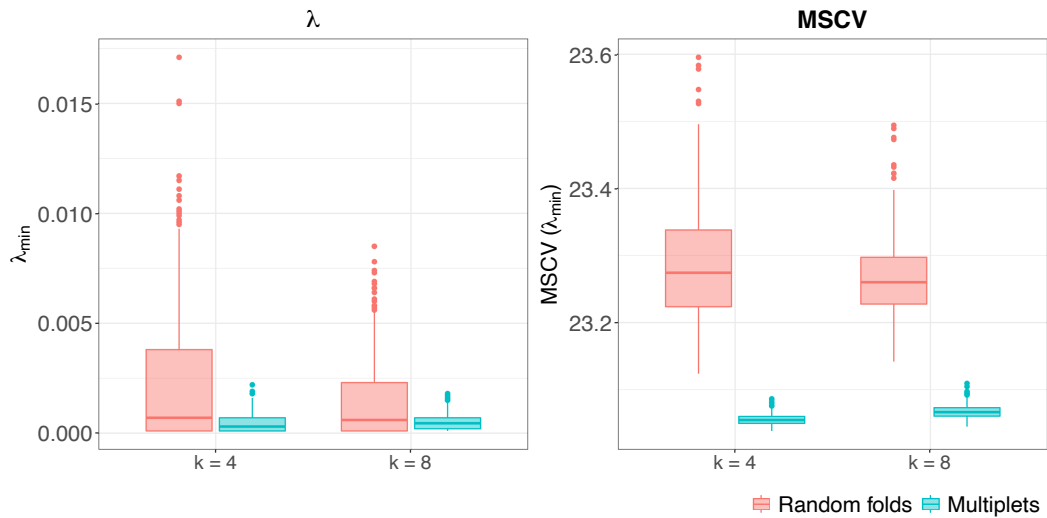


Figure 2.10: Distribution of λ_{min} (left) and MSCV (right) at λ_{min} , as estimated by LASSO with 4-fold and 8-fold cross validation, over 500 distinct random folds and multiplets of the airfoil self-noise dataset.

2.7 Conclusions

Twinning is built upon `SPlit` that is aimed at optimally splitting a dataset into training and testing sets. The existing implementation of `SPlit` uses difference-of-convex programming to find support points of the dataset, and then the testing set is sampled from the dataset using a sequential nearest neighbor algorithm (the overall procedure is termed as DC-NN in this chapter). This procedure can be time consuming, and thus, not applicable to even

moderately large datasets. On the other hand, `Twinning` directly minimizes the energy distance between the two twin sets, and is several orders of magnitude faster than `DC-NN`. This computational breakthrough allows `Twinning` to solve many problems that couldn't even be imagined with `SPLIT`.

We have described multiple potential applications of `Twinning` besides data splitting, such as data compression and cross-validation, and have demonstrated its performance on a few real datasets. For example, the taxicab dataset containing more than two million data points were split into two sets of 1.66 million and 0.41 million using `Twinning` in about two minutes on an ordinary desktop computer. This would not have been possible with `DC-NN`, which would have taken about a month to compute on a similar computer. This computational advantage is crucial for applications such as data splitting and data subsampling. If an optimal procedure for data subsampling takes more time than fitting a computationally expensive statistical model, then it does not make any sense to subsample the data to save model fitting time, one could rather fit the model directly on the full data. The speed at which `Twinning` can be executed is quite remarkable that it opens the door to a wide variety of problems to which `Twinning` can be applied, and we expect many more to be discovered in the future.

CHAPTER 3

A UNIFIED GLOBAL-LOCAL GAUSSIAN PROCESS APPROXIMATION

3.1 Introduction

Gaussian process (GP) is a widely used Bayesian framework for nonparametric regression (C. E. Rasmussen & Williams, 2005), and emulating computer models (Gramacy, 2020; Santner et al., 2003). The objective is to approximate a latent function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$, for which a functional Gaussian prior is assumed, and the posterior is obtained given the observed training data. The posterior mean is treated as the point prediction at \mathbf{x} , and the posterior variance gives the associated prediction uncertainty. Regardless of being the best linear unbiased predictor under the assumed model, a major drawback with GP modeling that limits its applicability to Big Data is its $O(n^3)$ computational complexity, where n is the number of observations in the training data. This is because GP modeling requires the inversion of an $n \times n$ kernel (or correlation) matrix \mathcal{R}_m involved in posterior estimation. Building GP approximations that scale reasonably well with large n is an active area of research, and the various methodologies to do so can be roughly grouped into two categories: (i) global, and (ii) local approximations.

Global approximations generally (i) focus on a subset of the training data with $m \ll n$ observations resulting in a much smaller \mathcal{R}_{mm} to invert (Chalupka et al., 2013), where the subset can be randomly selected, based on clustering, or with active learning (Keerthi & Chu, 2005; Lawrence et al., 2002), (ii) use a compactly supported kernel function (Gneiting, 2002) to generate sparse \mathcal{R}_m , then exploit the sparsity to efficiently compute \mathcal{R}_m^{-1} (Kaufman et al., 2011), or (iii) approximate \mathcal{R}_m with a low m -rank matrix plus diagonal using $m \ll n$ inducing points, that can be inverted in $O(m^2n)$ (Quiñonero-Candela & Rasmussen, 2005). Local approximations essentially (i) build a local GP model for each testing location (Gramacy &

Apley, 2015), or (ii) partition the training data into disjoint blocks and build independent GP for each block resulting in a block diagonal \mathcal{R}_m that can be inverted efficiently (Das & Srivastava, 2010).

Global or local approximations alone can be insufficient depending upon the data. To model a complex latent function $f(\mathbf{x})$ that varies significantly locally, a larger m would be needed to adequately summarize the data, be it subset of data approach or inducing points. Local approximations where a local GP is fit in the neighborhood of the testing location ignore the global trend and often result in over-confident predictions due to local over-fitting. On the other hand, local approximations where independent GPs are fit on partitioned training data suffer from discontinuity at the boundaries of the local regions. Park and Huang, 2016 show that greater the discontinuity lower the prediction accuracy, especially at the boundaries.

Snelson and Ghahramani, 2007 build on the inducing points framework presented in Quiñonero-Candela and Rasmussen, 2005 to incorporate local trend. To begin with, the inducing points framework makes the assumption that given the inducing points \mathcal{I} , the training and testing conditional distributions are independent, i.e, for any training location \mathbf{x} and testing location \mathbf{x}^* , $p(f(\mathbf{x}), f(\mathbf{x}^*)|\mathcal{I}) = p(f(\mathbf{x})|\mathcal{I})p(f(\mathbf{x}^*)|\mathcal{I})$. The fully independent conditional approximation (FIC) given in Snelson and Ghahramani, 2005 makes additional assumption that given \mathcal{I} , the conditional distribution of the latent function at any two locations (training and/or testing) $\mathbf{x}_i, \mathbf{x}_j$ are independent, i.e., $p(f(\mathbf{x}_i), f(\mathbf{x}_j)|\mathcal{I}) = p(f(\mathbf{x}_i)|\mathcal{I})p(f(\mathbf{x}_j)|\mathcal{I})$. Snelson and Ghahramani, 2007 later present partially independent conditional approximation (PIC) where the space is partitioned into disjoint blocks and given \mathcal{I} , the conditional distribution at any two locations are independent only when they belong to separate blocks. The dependence within a block in PIC attempts to capture the local trend, with the limitation that at boundaries of a block the dependence from locations in neighboring blocks are ignored.

The discontinuity problem alluded to before with local approximations that build independent GPs on partitioned training data is generally addressed by using some form of

weighted averaging of the independent GPs (T. Chen & Ren, 2009; Deisenroth & Ng, 2015; Gramacy & Lee, 2008; C. Rasmussen & Ghahramani, 2001; Tresp, 2000). Park and Apley, 2018 present a patching of the independent GPs (PK) where they augment the training data with a set of pseudo observations located at the boundaries of neighboring regions, and impose continuity by enforcing two neighboring GPs to make identical predictions at the pseudo locations common to them. Though the discontinuity problem can be accounted for, the global trend remains elusive for local approximations.

In this work we propose a novel methodology to capture both global and local trend in GP modeling. We first select a set of observations from the training data, independent of the testing location, to capture the global trend. The global point set is then supplemented with observations from the neighborhood of a given testing location to capture the local trend. The global trend is modeled using a power exponential kernel whose hyperparameters are learned based on the global points alone, and the local trend is modeled using a compactly supported kernel with a single hyperparameter that is predetermined based on the global points. Both kernels act on the combined set of global and local points of size $m \ll n$, resulting in an additive kernel. Vanhatalo and Vehtari, 2008 also use an additive kernel where the global trend is modeled with FIC and local trend with a compactly supported kernel, however, contrary to our approach, their compactly supported kernel acts on the full training data and requires the training data to have lattice structure to efficiently invert the kernel matrix by exploiting its sparsity. Our framework does not impose any restrictions on the training data, and scales well in higher dimensions as we do not rely on the sparsity of the compactly supported kernel matrix for efficient inversion. Furthermore, unlike FIC and PIC, we do not make any assumptions on the conditional distributions.

We refer to our methodology as `TwinGP` owing to the twin set of training points and kernels involved. The remainder of the article is organized as follows. Section 3.2 provides a brief review of GP and introduces notation. Section 3.3 presents the new `TwinGP` framework. Section 3.4 gives an illustration of `TwinGP` using a $1d$ function, and in Section 3.5 we

compare TwinGP with popular global, and local GP approximations for emulation as well as modeling real world datasets. Finally, in Section 3.6 we provide our concluding remarks.

3.2 Gaussian Process Review

Denote the training data as $\{\mathbf{X}_n, \mathbf{Y}_n\} := \{\mathbf{x}_i, y_i\}_{i=1}^n$ such that the inputs $\mathbf{x}_i \in \mathbb{R}^d$ and output $y_i \in \mathbb{R}$, for all i . Let

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \text{ for } i = 1, \dots, n, \quad (3.1)$$

where $\epsilon_i \stackrel{iid}{\sim} \mathcal{N}(0, \nu^2)$ is the random noise in the output. Our aim is to estimate the latent function $f(\cdot)$ from the training data. In GP modeling, we assume that $f(\cdot)$ to be a realization from a Gaussian process:

$$f(\mathbf{x}) \sim \text{GP}(\mu, \tau^2 \mathcal{R}(\mathbf{x}, \cdot)), \quad (3.2)$$

where μ , τ^2 , and $\mathcal{R}(\cdot, \cdot)$ are the mean, variance, and correlation function of the GP, respectively. The correlation function is defined as $\text{Cor}\{f(\mathbf{u}), f(\mathbf{v})\} = \mathcal{R}(\mathbf{u}, \mathbf{v})$, which is a positive definite function with $\mathcal{R}(\mathbf{u}, \mathbf{u}) = 1$. We use the names correlation function and kernel function interchangeably in this paper. Please refer to the books Santner et al., 2003 and Gramacy, 2020 for details on GP modeling.

From (3.1) and (3.2), we have $\mathbf{Y}_n \sim \mathcal{N}_n(\mu \mathbf{1}_n, \tau^2 \mathcal{R}_{nn} + \nu^2 \mathbf{I}_n)$, where \mathcal{R}_{nn} is the $n \times n$ correlation matrix whose ij^{th} element is $\mathcal{R}(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{1}_n := [1, \dots, 1]'$, and \mathbf{I}_n is the $n \times n$ identity matrix. Let $\eta = \nu^2 / \tau^2$, known as nugget. Given an arbitrary testing location $\mathbf{x}^* \in \mathbb{R}^d$, we are interested in finding the conditional distribution of $f(\mathbf{x}^*) | \mathbf{Y}_n$.

Define the $1 \times n$ correlation vector as $\mathcal{R}(\mathbf{x}^*, \mathbf{X}_n) := [\mathcal{R}(\mathbf{x}^*, \mathbf{x}_1), \dots, \mathcal{R}(\mathbf{x}^*, \mathbf{x}_n)]$. In keeping with the notation, we have $\mathcal{R}(\mathbf{X}_n, \mathbf{x}^*) = \mathcal{R}(\mathbf{x}^*, \mathbf{X}_n)'$ and $\mathcal{R}_{nn} = \mathcal{R}(\mathbf{X}_n, \mathbf{X}_n) := [\mathcal{R}(\mathbf{x}_1, \mathbf{x}_1); \dots; \mathcal{R}(\mathbf{x}_n, \mathbf{x}_n)]$. The joint distribution of $f(\mathbf{x}^*)$ and $\mathbf{f}(\mathbf{X}_n) := [f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)]'$ is given by

$$\begin{bmatrix} f(\mathbf{x}^*) \\ \mathbf{f}(\mathbf{X}_n) \end{bmatrix} \sim \mathcal{N}_{n+1} \left(\begin{bmatrix} \mu \\ \mu \mathbf{1}_n \end{bmatrix}, \tau^2 \begin{bmatrix} 1 & \mathcal{R}(\mathbf{x}^*, \mathbf{X}_n) \\ \mathcal{R}(\mathbf{X}_n, \mathbf{x}^*) & \mathcal{R}_{nn} + \eta \mathbf{I}_n \end{bmatrix} \right).$$

Then, the conditional distribution of $f(\mathbf{x}^*)|\mathbf{Y}_n$, is given by

$$f(\mathbf{x}^*)|\mathbf{Y}_n \sim \mathcal{N}(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*)),$$

where

$$\mu(\mathbf{x}^*) = \mu + \mathcal{R}(\mathbf{x}^*, \mathbf{X}_n)[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}(\mathbf{Y}_n - \mu\mathbf{1}_n), \quad (3.3)$$

$$\sigma^2(\mathbf{x}^*) = \tau^2 \{1 - \mathcal{R}(\mathbf{x}^*, \mathbf{X}_n)[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}\mathcal{R}(\mathbf{X}_n, \mathbf{x}^*)\}. \quad (3.4)$$

There exists a multitude of correlation functions in the literature (C. E. Rasmussen & Williams, 2005, Ch.4), each with their own set of correlation parameters θ that are estimated from the training data. The empirical Bayes estimate of μ , τ^2 , η , and θ are given as follows (Santner et al., 2003):

$$\hat{\mu} = \frac{\mathbf{1}_n'[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}\mathbf{Y}_n}{\mathbf{1}_n'[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}\mathbf{1}_n}, \quad (3.5)$$

$$\hat{\tau}^2 = \frac{1}{n}(\mathbf{Y}_n - \hat{\mu}\mathbf{1}_n)'[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}(\mathbf{Y}_n - \hat{\mu}\mathbf{1}_n), \quad (3.6)$$

$$\hat{\theta}, \hat{\eta} = \arg \min_{\theta, \eta > 0} n \log \hat{\tau}^2 + \log |\mathcal{R}_{nn} + \eta\mathbf{I}_n|. \quad (3.7)$$

As evident from the above equations, much of the complexity in GP modeling stems from the computation of $[\mathcal{R}_{nn} + \eta\mathbf{I}_n]^{-1}$ and $|\mathcal{R}_{nn} + \eta\mathbf{I}_n|$, both requiring $O(n^3)$ operations. In the next section, we propose our methodology to address this computational bottleneck for large n .

3.3 Global-Local Gaussian Process

In order to circumvent the $O(n^3)$ computational complexity, we build the GP using only $m \ll n$ points from the training data. Contrary to the global approximation methodologies in the literature where the m points are chosen from the training data or artificially generated, so as to summarize the whole data, we use a combination of g global points and l local

points such that $m = g + l$. The g global points are independent of the testing location, while the l local points are specific to the testing location \mathbf{x}^* .

To capture the global trend, and local trend with respect to \mathbf{x}^* , the kernel function $\mathcal{R}(\cdot, \cdot)$ is also modeled as a combination of two kernel functions (Ba & Joseph, 2012; Harari & Steinberg, 2014), i.e.,

$$\mathcal{R}(\mathbf{x}_a, \mathbf{x}_b) = (1 - \lambda)\mathcal{G}(\mathbf{x}_a, \mathbf{x}_b) + \lambda\mathcal{L}(\mathbf{x}_a, \mathbf{x}_b), \quad \lambda \in [0, 1]. \quad (3.8)$$

\mathcal{G} is designed to capture the global trend in the data, while \mathcal{L} captures the local trend around \mathbf{x}^* . The hyperparameter λ controls the proportion of global and local trends used in building \mathcal{R} .

Let $\mathbf{X}_m \subset \mathbf{X}_n$ represent the m training points, then $\mathcal{G}_{mm} = \mathcal{G}(\mathbf{X}_m, \mathbf{X}_m)$, and $\mathcal{L}_{mm} = \mathcal{L}(\mathbf{X}_m, \mathbf{X}_m)$. With this setup, instead of inverting $\mathcal{R}_{nn} + \eta\mathbf{I}_n$ in (3.3)-(3.7), we need only invert $\mathcal{R}_{mm} + \eta\mathbf{I}_m$ where

$$\mathcal{R}_{mm} = (1 - \lambda)\mathcal{G}_{mm} + \lambda\mathcal{L}_{mm}. \quad (3.9)$$

Thus, the computational complexity of fitting the GP reduces to $O(m^3)$. In the subsections that follow, we present how to efficiently locate the m training points, the final predictive equations, and how the kernel function $\mathcal{R}(\cdot, \cdot)$ in (3.8) is determined.

3.3.1 Global and Local Points

Denote the global training points as \mathbf{X}_g , and the local training points as \mathbf{X}_l , we have $\mathbf{X}_m = \mathbf{X}_g \cup \mathbf{X}_l$. Ideally, for a given testing location \mathbf{x}^* , we should be denoting \mathbf{X}_m as $\mathbf{X}_m(\mathbf{x}^*) = \mathbf{X}_g \cup \mathbf{X}_l(\mathbf{x}^*)$, a technicality we omit for ease of notation.

The objective we desire to achieve with \mathbf{X}_g is to sufficiently model the global trend in the training data. Mak and Joseph, 2018a proposed a nonparametric and model-free data reduction technique known as support points that can produce a small set of points to

represent the full dataset. Support points are obtained by minimizing the energy distance (Székely & Rizzo, 2013) between its empirical distribution and that of the dataset using difference-of-convex programming. However, support points need not be a subset of the full dataset. Given the support points, `SPLIT` uses sequential nearest neighbor assignment to sample from the dataset. Even still, computing support points as given in Mak and Joseph, 2018a is $O(n^2)$. `Twinning` is an efficient sampling algorithm that minimizes the energy distance in $O(n \log n)$. Since we are dealing with large datasets in this work, we will use `Twinning` to find \mathbf{X}_g .

Given the testing location \mathbf{x}^* , \mathbf{X}_l is obtained as the l nearest neighbors to \mathbf{x}^* in $\mathbf{X}_n \setminus \mathbf{X}_g$, a task that can be performed efficiently with kd -tree in $O(l \log l)$. Figure 3.1 gives a depiction of the global and local training points identified for a given testing location, for a $1d$ example. One can observe from the figure that fitting a GP on the local points alone, shown as green diamonds, will clearly underpredict at the testing location, and it is shown to be not optimal (Emery, 2009). In `1aGP` proposed by Gramacy and Apley, 2015, the nearest neighbors are supplemented with active learning, however, an optimization is required at each testing location that adds to the computational complexity. On the other hand, our method makes use of the predetermined global points (blue circles) to capture the global trend, and therefore, nearest neighbors alone as the local points will suffice. This reduces the computational burden in our framework.

3.3.2 Predictive equations

At this stage, given \mathbf{x}^* , we have identified \mathbf{X}_m . Assume that the kernel function \mathcal{R} is fully determined. Let \mathbf{Y}_m be the response vector corresponding to \mathbf{X}_m , i.e., $\mathbf{Y}_m = [\mathbf{Y}_g; \mathbf{Y}_l]$ where \mathbf{Y}_g and \mathbf{Y}_l are the response vectors corresponding to \mathbf{X}_g and \mathbf{X}_l , respectively. The conditional distribution of $f(\mathbf{x}^*)|\mathbf{Y}_m$ is given by $\mathcal{N}(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*))$, where

$$\mu(\mathbf{x}^*) = \hat{\mu}_m + \mathcal{R}(\mathbf{x}^*, \mathbf{X}_m)[\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1}(\mathbf{Y}_m - \mu \mathbf{1}_m), \quad (3.10)$$

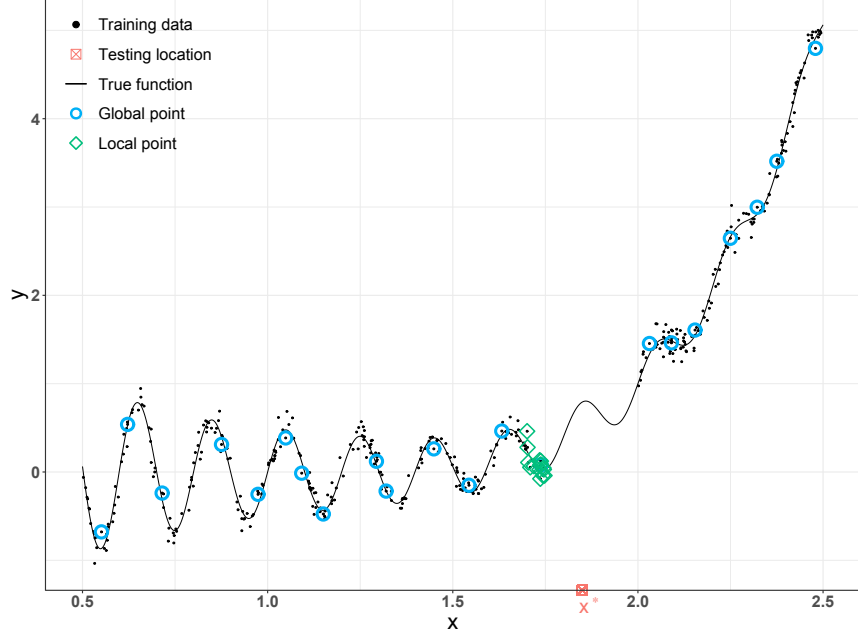


Figure 3.1: Illustration of the training points (global and local) identified for a given training data and testing location with TwinGP.

$$\sigma^2(\mathbf{x}^*) = \hat{\tau}_m^2 \left\{ 1 - \mathcal{R}(\mathbf{x}^*, \mathbf{X}_m) [\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1} \mathcal{R}(\mathbf{X}_m, \mathbf{x}^*) \right\}, \quad (3.11)$$

$$\hat{\mu}_m = \frac{\mathbf{1}_m' [\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1} \mathbf{Y}_m}{\mathbf{1}_m' [\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1} \mathbf{1}_m}, \quad (3.12)$$

$$\hat{\tau}_m^2 = \frac{1}{m} (\mathbf{Y}_m - \hat{\mu}_m \mathbf{1}_m)' [\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1} (\mathbf{Y}_m - \hat{\mu}_m \mathbf{1}_m). \quad (3.13)$$

For the prediction of a noisy observation, we have $y^* | \mathbf{Y}_m \sim \mathcal{N}(\mu(\mathbf{x}^*), \sigma_\eta^2(\mathbf{x}^*))$, where $\mu(\mathbf{x}^*)$ is the same as given in (3.10), but $\sigma_\eta^2(\mathbf{x}^*)$ changes to

$$\sigma_\eta^2(\mathbf{x}^*) = \hat{\tau}_m^2 \left\{ 1 + \eta - \mathcal{R}(\mathbf{x}^*, \mathbf{X}_m) [\mathcal{R}_{mm} + \eta \mathbf{I}_m]^{-1} \mathcal{R}(\mathbf{X}_m, \mathbf{x}^*) \right\}. \quad (3.14)$$

Furthermore, $\mathcal{R}_{mm} + \eta \mathbf{I}_m$ can be deconstructed as follows,

$$\mathcal{R}_{mm} + \eta \mathbf{I}_m = \begin{bmatrix} \mathcal{R}_{gg} + \eta \mathbf{I}_g & \mathcal{R}_{gl} \\ \mathcal{R}_{lg} & \mathcal{R}_{ll} + \eta \mathbf{I}_l \end{bmatrix} \quad (3.15)$$

$$= \begin{bmatrix} (1-\lambda)\mathcal{G}_{gg} + \lambda\mathcal{L}_{gg} + \eta\mathbf{I}_g & (1-\lambda)\mathcal{G}_{gl} + \lambda\mathcal{L}_{gl} \\ (1-\lambda)\mathcal{G}_{lg} + \lambda\mathcal{L}_{lg} & (1-\lambda)\mathcal{G}_{ll} + \lambda\mathcal{L}_{ll} + \eta\mathbf{I}_l \end{bmatrix}. \quad (3.16)$$

Since $\mathcal{R}_{gg} = (1-\lambda)\mathcal{G}_{gg} + \lambda\mathcal{L}_{gg}$ is independent of the testing location \mathbf{x}^* , \mathcal{R}_{gg} and $[\mathcal{R}_{gg} + \eta\mathbf{I}_g]^{-1}$ can be predetermined before testing. For a given \mathbf{x}^* , the computational effort in inverting $\mathcal{R}_{mm} + \eta\mathbf{I}_m$ can be significantly reduced by using block matrix inversion as follows,

$$[\mathcal{R}_{mm} + \eta\mathbf{I}_m]^{-1} = \begin{bmatrix} \Sigma_{gg}^{-1} + \Sigma_{gg}^{-1}\mathcal{R}_{gl}\mathbf{S}^{-1}\mathcal{R}_{lg}\Sigma_{gg}^{-1} & -\Sigma_{gg}^{-1}\mathcal{R}_{gl}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathcal{R}_{lg}\Sigma_{gg}^{-1} & \mathbf{S}^{-1} \end{bmatrix}, \quad (3.17)$$

where $\Sigma_{gg} = \mathcal{R}_{gg} + \eta\mathbf{I}_g$, and $\mathbf{S} = \mathcal{R}_{ll} - \mathcal{R}_{lg}\Sigma_{gg}^{-1}\mathcal{R}_{gl}$, thereby, per testing location we need only invert \mathbf{S} , a small $l \times l$ matrix.

3.3.3 Correlation Functions

To model the global trend, we use the popular power exponential kernel function (Sacks et al., 1989),

$$\mathcal{G}(\mathbf{x}_a, \mathbf{x}_b) = \exp\left(-\sum_{i=1}^d \frac{|\mathbf{x}_a^i - \mathbf{x}_b^i|^\alpha}{\theta_g^i}\right), \quad \alpha \in (0, 2], \theta_g^i > 0. \quad (3.18)$$

Denote the hyperparameters of \mathcal{G} as $\boldsymbol{\theta}_g$, we have $\boldsymbol{\theta}_g = \{\theta_g^1, \dots, \theta_g^d, \alpha\}$. They can be estimated with respect to \mathbf{X}_g alone, i.e., independent of the testing location,

$$\hat{\mu}_g = \frac{\mathbf{1}_g'[\mathcal{G}_{gg} + \eta_g\mathbf{I}_g]^{-1}\mathbf{Y}_g}{\mathbf{1}_g'[\mathcal{G}_{gg} + \eta_g\mathbf{I}_g]^{-1}\mathbf{1}_g}, \quad (3.19)$$

$$\hat{\tau}_g^2 = \frac{1}{g}(\mathbf{Y}_g - \hat{\mu}_g\mathbf{1}_g)'[\mathcal{G}_{gg} + \eta_g\mathbf{I}_g]^{-1}(\mathbf{Y}_g - \hat{\mu}_g\mathbf{1}_g), \quad (3.20)$$

$$\hat{\boldsymbol{\theta}}_g = \arg \min_{\boldsymbol{\theta}_g} g \log \hat{\tau}_g^2 + \log |\mathcal{G}_{gg} + \eta_g\mathbf{I}_g|. \quad (3.21)$$

We have constrained $\alpha \in [1, 2]$ in the optimization assuming the latent function $f(\cdot)$ to be reasonably smooth and not too wiggly.

To model the local trend, we use compactly supported correlation functions (Gneiting, 2002), which ensures identifiability of the local correlation parameters with respect to the global correlation parameters. Specifically, we use Wendlands's compactly supported radial function (Wendland, 2004),

$$\mathcal{L}(\mathbf{x}_a, \mathbf{x}_b) = \left((q+1) \frac{\|\mathbf{x}_a - \mathbf{x}_b\|_2}{\theta_l} + 1 \right) \max \left\{ 0, \left(1 - \frac{\|\mathbf{x}_a - \mathbf{x}_b\|_2}{\theta_l} \right) \right\}^{q+1}, \quad q = \lfloor \frac{d}{2} \rfloor + 2 \quad (3.22)$$

where $\|\cdot\|_2$ is the ℓ_2 norm and $\lfloor u \rfloor$ is the largest integer lesser than or equal to u . \mathcal{L} has a single hyperparameter θ_l which we set as the covering radius (Fasshauer, 2007, p. 22) for \mathbf{X}_g to cover \mathbf{X}_n , i.e.,

$$\hat{\theta}_l = \min \left\{ \rho : \mathbf{X}_n \subseteq \cup_{i=1}^g \mathcal{B}_\rho(\mathbf{x}_i), \mathbf{x}_i \in \mathbf{X}_g, \forall i \right\}, \quad (3.23)$$

where $\mathcal{B}_\rho(\mathbf{x})$ is a closed ball of radius ρ centered at \mathbf{x} . Setting θ_l as in (3.23) makes it independent of the testing location, thereby, allowing us to precompute \mathcal{L}_{gg} in (3.16) leading to the computational gains with block matrix inversion (3.17). In addition, almost surely for any testing location \mathbf{x}^* there exists at least one global training point such that the local kernel is active between them, i.e., there exists $\mathbf{x}_g \in \mathbf{X}_g : \mathcal{L}(\mathbf{x}^*, \mathbf{x}_g) > 0$. The motivation here is that we do not wish \mathcal{L} to neglect the correlation between \mathbf{x}^* and \mathbf{X}_g .

We need to specify two more parameters: λ and η_l . We could estimate them using empirical Bayes methods as before, but since we have unused data in the training set, we can do the estimation in a different and more robust fashion. We sample $\{\mathbf{X}_v, \mathbf{Y}_v\}$ from $\{\mathbf{X}_n, \mathbf{Y}_n\} \setminus \{\mathbf{X}_g, \mathbf{Y}_g\}$ by Twinning to create a validation set. Now, we can estimate λ and η_l by minimizing the prediction error:

$$\begin{aligned} \hat{\lambda}, \hat{\eta}_l &= \arg \min_{\lambda \in [0,1], \eta_l > 0} \text{MSE}(\lambda) \\ &= \arg \min_{\lambda \in [0,1], \eta_l > 0} \sum_{\mathbf{x} \in \mathbf{X}_v} (y_{\mathbf{x}} - \mu(\mathbf{x}))^2, \end{aligned} \quad (3.24)$$

where $\mu(\mathbf{x})$ is obtained as given in (3.10) with $\eta = (1 - \lambda)\eta_g + \lambda\eta_l$.

3.3.4 Computational complexity

The TwinGP procedure is summarized in Algorithm 3. The computational complexity of TwinGP can be deconstructed as follows,

Testing

- (i) Identifying \mathbf{X}_l for a given \mathbf{x}^* is on average $O(\log n)$ using *kd*-tree.
- (ii) The complexity in computing $\mu(\mathbf{x}^*)$ and $\sigma^2(\mathbf{x}^*)$ is dictated by inversion of \mathcal{R}_{mm} . With block matrix inversion as given in (3.17), computing $[\mathcal{R}_{mm} + \eta\mathbf{I}_m]^{-1}$ is $O(g^2l)$.

Training

- (i) Obtaining a twinning sample to identify \mathbf{X}_g is on average $O(n \log n)$.
- (ii) The complexity in estimating θ_g is dictated by inversion of $\mathcal{G}_{gg} + \eta_g\mathbf{I}_g$ that is $O(g^3)$.
- (iii) θ_l can be estimated efficiently with a *kd*-tree in $O(n \log g)$.
- (iv) The complexity in estimating λ and η_l is similar to testing, i.e., $O(g^2l)$ per validation point.

Overall complexity

Thus, the overall computational complexity of TwinGP is $O(g^3 + tg^2l)$, where t is the number of testing locations. If we select g to be order of \sqrt{n} , the training complexity reduces to $O(n^{1.5})$, a substantial improvement over $O(n^3)$.

Algorithm 3 TwinGP

- 1: Input training set $\{\mathbf{X}_n, \mathbf{Y}_n\}$ and testing locations \mathbf{X}_t^*
 - 2: Identify global training points \mathbf{X}_g (Section 3.3.1)
 - 3: Estimate $\boldsymbol{\theta}_g$ as in (3.21)
 - 4: Estimate θ_l as in (3.23)
 - 5: Estimate λ and η_l as in (3.24)
 - 6: **for** $\mathbf{x}^* \in \mathbf{X}_t^*$ **do**
 - 7: Identify local training points \mathbf{X}_l (Section 3.3.1)
 - 8: Compute $\mu(\mathbf{x}^*)$ and $\sigma^2(\mathbf{x}^*)$ (Section 3.3.2)
 - 9: **end for**
 - 10: **return** $\{\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*)\}, \forall \mathbf{x}^* \in \mathbf{X}_t^*$
-

3.4 1d Illustration

Here we illustrate how TwinGP overcomes the shortcomings of purely local or global GP approximations. Consider the following function from Gramacy and Lee, 2012,

$$f(x) = \frac{\sin 10\pi x}{2x} + (x - 1)^4, \quad x \in [0.5, 2.5]. \quad (3.25)$$

We first generate the training dataset with $n = 500$ observations, where x_1, \dots, x_n are selected on a uniform grid in $[0.5, 2.5]$, and we add a Gaussian noise to each observation as follows,

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, 0.01), \quad \forall i = 1, \dots, 500. \quad (3.26)$$

The testing set at 2,000 locations are also selected on a uniform grid in $[0.5, 2.5]$. Plot (a) in Figure 3.2 depicts $f(x)$, the 500 training locations, and the prediction from a full GP modeled using the `mlegp` (Dancik & Dorman, 2008) package. As evident, $f(x)$ is recovered extremely well with a full GP, in addition, the confidence intervals are tight.

To demonstrate the global GP approximation method, we use SGPR (Titsias, 2009) implemented in `GPYtorch` (Gardner et al., 2018). SGPR is a low-rank GP approximation where the inducing points and kernel hyperparameters are estimated with a variational learning approach. For local approximation we use `laGP` (Gramacy & Apley, 2015) implemented

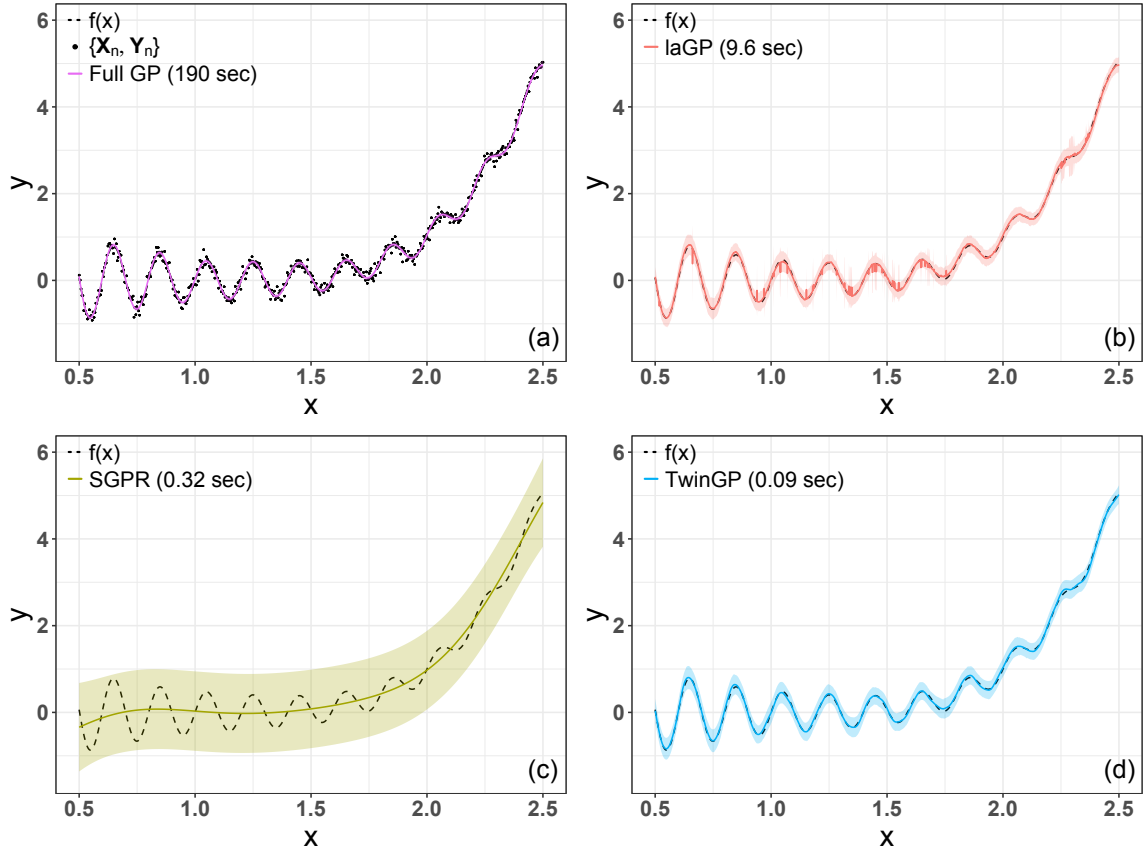


Figure 3.2: Prediction and 2σ confidence interval (shaded region) with full GP, laGP, SGPR, and TwinGP to model the 1d function in (3.25).

in the laGP (Gramacy, 2016) package. For each testing location, laGP starts with a set of training points in its neighborhood and then sequentially adds more training points so as to minimize the predictive variance.

For TwinGP we set the number of global points $g = 22$ and local points $l = 25$. For SGPR the number of inducing points is set to be $m = g + l$, and for laGP $l + 10$ training locations are considered per testing location, i.e., start with l neighborhood points and then sequentially add 10 more points. Plots (b), (c), and (d) in Figure 3.2 give the predictions and 2σ confidence intervals from laGP, SGPR, and TwinGP respectively. As expected, laGP predictions are discontinuous owing to its local nature, while SGPR produces smooth prediction neglecting the local oscillations of $f(x)$. On the other hand, TwinGP gives comparable predictions to the full GP. The total computation time for training and testing

with the four methods are given in Figure 3.2. We can clearly see the computational saving with TwinGP over the full GP, which will be even more substantial with large datasets as we demonstrate in the next section.

3.5 Experiments

In this section, we make an extensive analysis of TwinGP performance on several emulation, and real world datasets, compared to other scalable GP modeling frameworks. Similar to Section 3.4, in our experiments we include SGPR for global approximation, laGP for local approximation, and patchwork kriging (PK) by Park and Apley, 2018 introduced in Section 3.1. All the experiments presented in this section are carried out on a 2.6 GHz 6-Core Intel i7-9750H processor with 16 GB memory. We apply the following general settings for the different modeling frameworks, unless stated otherwise.

TwinGP: The number of global points g is set as $\min\{50d, \max\{\lfloor \sqrt{n} \rfloor, 10d\}\}$, i.e., at least $10d$ points are chosen to model the global trend, with an upper bound of $50d$. The local trend is modeled with $l = \max\{25, 3d\}$ points. The number of validation points used for estimating λ and η is set as $2g$. The θ_g hyperparameter optimization is done with grid initialization and multi-starts as outlined in Basak et al., 2022.

SGPR: For a fair comparison with TwinGP, we set the number of inducing points to be $m = g + l$. We follow the implementation provided in GPyTorch documentation¹. The number of iterations in hyperparameter optimization is modified from constant 100 to $\min\{250, \lfloor 50 \log(1 + d) \rfloor\}$, and a learning rate of 0.05 is used instead of 0.01. In addition, separate lengthscales are learnt per dimension.

laGP: For each testing location, we start with l training points in its neighborhood and sequentially add 10 more points to the design that minimize the predictive variance.

¹https://docs.gpytorch.ai/en/latest/examples/02_Scalable_Exact_GPs/SGPR_Regression_CUDA.html

The aGPsep function provided in the R package by Gramacy, 2016 is used to execute laGP in parallel. For emulation datasets where there is no observation noise, the nugget parameter is set to 10^{-7} , and for real world datasets the nugget is set to NULL in which case it is estimated.

PK: The training locations are partitioned into K disjoint blocks such that each block contains at least $10d$ points. The spatial tree algorithm used to make the partition benefits from K being a power of 2, hence, we set $K = 2^{\lfloor \log_2 \frac{n}{10d} \rfloor}$. The number of pseudo observations B introduced at the boundaries of neighboring blocks to enforce continuity between the neighboring GPs is set to be 3. Our choice of B is conservative as the complexity of PK scales proportionally to B^3 . We use the MATLAB implementation of PK provided by the author².

3.5.1 Evaluation criteria

The performance of the different modeling frameworks is assessed based on their prediction accuracy and total computation time. The prediction accuracy is quantified with root mean squared error (RMSE) and negative log predictive density (NLPD). RMSE measures the quality of point predictions from the model, while NLPD measures how well the predicted posterior distribution fits the testing data. Lower the RMSE and NLPD values, better the performance of the model. Given the testing set $\{\mathbf{X}_t^*, \mathbf{Y}_t^*\}$ with t testing locations, we have

$$\text{RMSE} = \sqrt{\frac{1}{t} \sum_{i=1}^t \{y_i^* - \mu(\mathbf{x}_i^*)\}^2}, \quad (3.27)$$

$$\text{NLPD} = \frac{1}{2t} \sum_{i=1}^t \left[\frac{\{y_i^* - \mu(\mathbf{x}_i^*)\}^2}{\sigma^2(\mathbf{x}_i^*)} + \log\{2\pi\sigma^2(\mathbf{x}_i^*)\} \right]. \quad (3.28)$$

²<https://www.chiwoopark.net/code-and-dataset>

3.5.2 Emulation

We consider three emulation problems here, the piston simulation function (Kenett & Zacks, 2021), the borehole function (Morris et al., 1993), and the Dette & Pepelyshev function (Dette & Pepelyshev, 2010). For a given emulation experiment, the input dimensions are scaled to $[0, 1]$ and $n = 10,000$ training locations are sampled uniformly from a d -dimensional hypercube, i.e., $\mathbf{X}_n \sim \text{Unif}(0, 1)^{n \times d}$, and 2,000 deterministic Sobol sequence in d dimensions are the testing locations. A GP is modeled on the training locations using TwinGP, SGPR, laGP, and PK. The performance of the fitted models is assessed based on the criteria given in Section 3.5.1. The procedure is repeated for 50 iterations with different training locations sampled similarly, while the testing locations remain unchanged. The distribution of RMSE and NLPD over the 50 iterations are presented as box-and-whisker plots. Further information and code for the emulations problems can be obtained from Simon Fraser University’s virtual library of simulation experiments³.

Piston simulation function

The piston simulation function models the circular motion of a piston within a cylinder (Kenett & Zacks, 2021). There are $d = 7$ inputs, which are piston weight, piston surface area, initial gas volume, spring coefficient, atmospheric pressure, ambient temperature, and filling gas temperature. The response is the time taken to complete one cycle in seconds. The experiment results are given in Figure 3.3. We see that TwinGP performs the best in terms of RMSE, and for NLPD, on average, its performance is comparable to that of PK. Computation time is the least for TwinGP, around 2 seconds, while PK took around 80 seconds.

³<https://www.sfu.ca/~ssurjano/emulat.html>

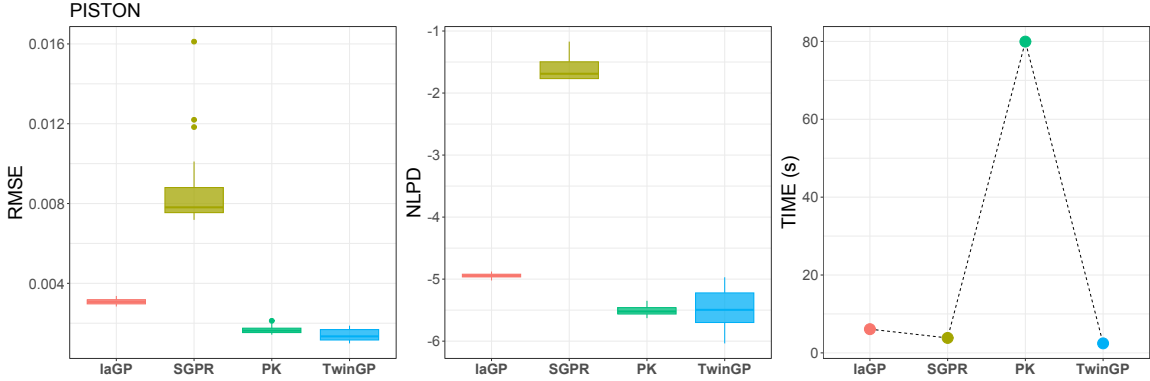


Figure 3.3: Distribution of RMSE and NLPD over 50 iterations of modeling the piston simulation function with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.

Borehole function

The borehole function models water flow through a borehole (Morris et al., 1993). There are $d = 8$ inputs, which are radius of the borehole, radius of influence, transmissivity of upper aquifer, potentiometric head of upper aquifer, transmissivity of lower aquifer, potentiometric head of lower aquifer, length of borehole, and hydraulic conductivity of borehole. The response is the water flow rate in m^3/yr . The experiment results are given in Figure 3.4. We see that PK is the best performing modeling framework in terms of both RMSE and NLPD, with TwinGP being the close second. However, in terms of computation time, TwinGP performance is far superior.

Dette-Pepelyshev function

The Dette-Pepelyshev function from Dette and Pepelyshev, 2010 with $d = 8$ input variables is heavily curved in some variables, and less so in others. Following the general settings described in the beginning of Section 3.5, running PK with $K = 64$ encountered numerical instabilities, hence, we increased K to 128. The experiment results are given in figure 3.5. In terms of RMSE, both TwinGP and PK perform the best with comparable results. PK performance is the best when it comes to NLPD, though at a very high computational cost.

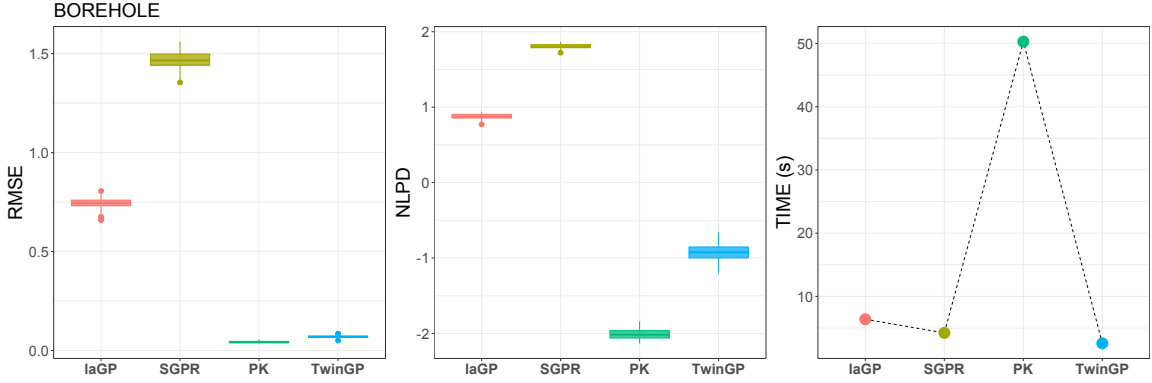


Figure 3.4: Distribution of RMSE and NLPD over 50 iterations of modeling the borehole function with `laGP`, `SGPR`, `PK`, and `TwinGP`. The corresponding average computation time per iteration is given in the right most plot.

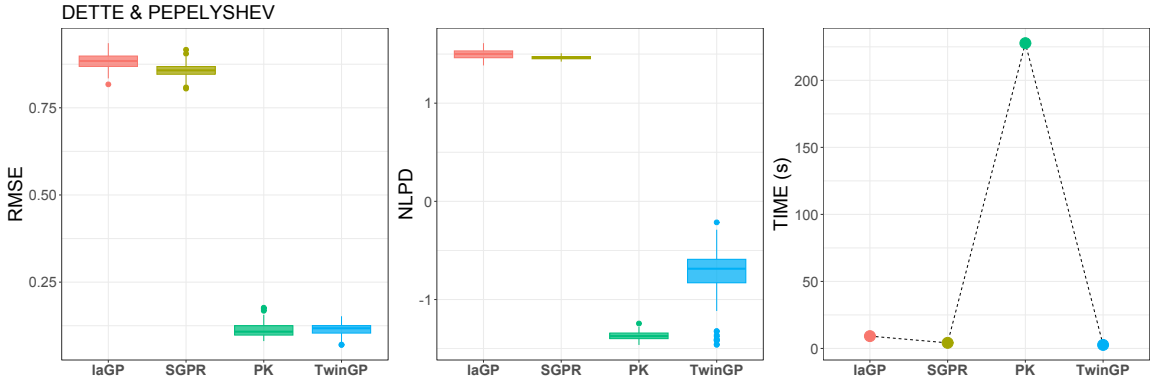


Figure 3.5: Distribution of RMSE and NLPD over 50 iterations of modeling the Dette-Pepelyshev function with `laGP`, `SGPR`, `PK`, and `TwinGP`. The corresponding average computation time per iteration is given in the right most plot.

3.5.3 Real world data

We consider four real world datasets here: the ozone column spatial dataset, protein tertiary structure dataset, Sarcos robotics dataset, and the flight delays dataset. The same datasets are considered in Park and Apley, 2018. For a given dataset, we first randomly split the dataset in 90-10 proportion and a GP is modeled with `TwinGP`, `SGPR`, `laGP`, and `PK` on 90% of the dataset, and the remaining 10% is used for testing the performance of the fitted models. The experiment is repeated for 50 iterations using different random splits. The distribution of RMSE and NLPD over the 50 iterations are presented as box-and-whisker plots.

Ozone column

The ozone column dataset contains measurement of total column of ozone by the NIMBUS-7/TOMS satellite on October 1, 1988, at different latitudes and longitudes over the globe. There are 182,591 observations with $d = 2$ inputs, latitude and longitude. The response is the total column of ozone. The experiment results are given in Figure 3.6. We omit SGPR in NLPD plot since it produced negative variances. In terms of RMSE, both TwinGP and PK provide the best performance. For NLPD, TwinGP on average performs better than PK, though PK is more consistent. Furthermore, TwinGP computation time is only 5 seconds, and is the fastest compared to the rest, while PK is the slowest at around 208 seconds. We used $K = 1024$ and $B = 3$ for PK, following the settings described at the beginning of Section 3.5. Decreasing K to 512 reduced the computation time to about 150 seconds with near identical RMSE and NLPD performance, even still, PK remained the slowest amongst rest of the models. Further decreasing K or increasing B for PK was computationally more expensive with no significant difference in RMSE and NLPD.

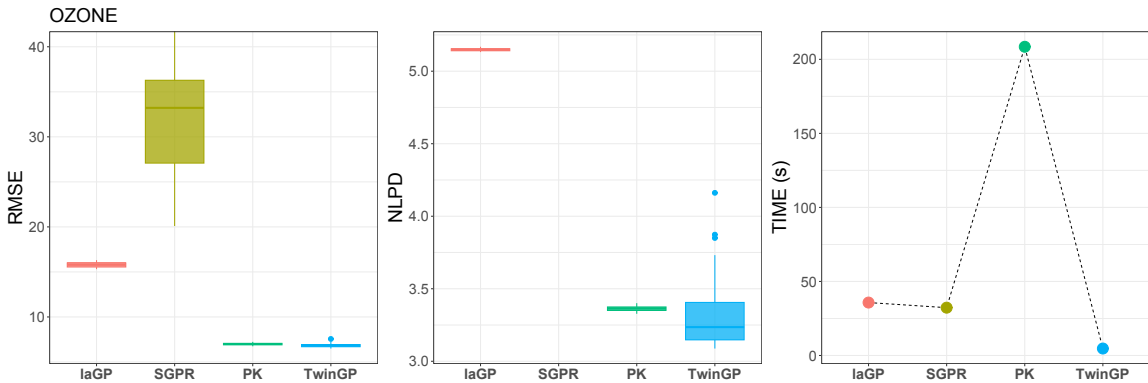


Figure 3.6: Distribution of RMSE and NLPD over 50 iterations of modeling the ozone column dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.

Protein tertiary structure

The protein tertiary structure dataset can be obtained from the UCI machine learning repository⁴. The dataset has $d = 9$ input variables relating to the physiochemical properties of protein tertiary structure, and the response is size of the protein residue. There are 45,730 observations in total. The experiment results are given in Figure 3.7. TwinGP is the best performing model in terms of RMSE. For NLPD, both TwinGP and PK performance is comparable, and are better than the rest. Though laGP is the fastest at around 13 seconds with TwinGP being the close second at about 17 seconds, laGP is the worst performing model in terms of RMSE.

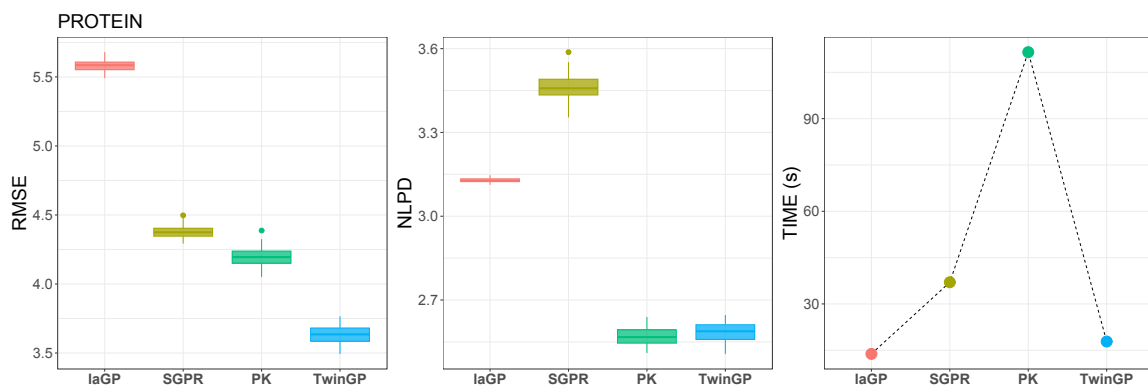


Figure 3.7: Distribution of RMSE and NLPD over 50 iterations of modeling the protein structure dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.

Sarcos robotics

The Sarcos robotics dataset⁵ (Vijayakumar & Schaal, 2000) has $d = 21$ input variables representing positions, velocities, and accelerations of a seven degrees-of-freedom Sarcos anthropomorphic robot arm, and there are 7 responses corresponding to the 7 joint torques. Similar to Park and Apley, 2018, we only consider the first response in modeling. The

⁴<https://archive.ics.uci.edu/ml/datasets>

⁵<http://gaussianprocess.org/gpml/data>

dataset is provided as training and testing sets which we combine, resulting in 48,933 total observations, before making the 90-10 random splits. The experiment results are given in Figure 3.8. Here we see that TwinGP is the best performing model in terms of all the evaluation criteria. TwinGP computation time is only about 47 seconds.

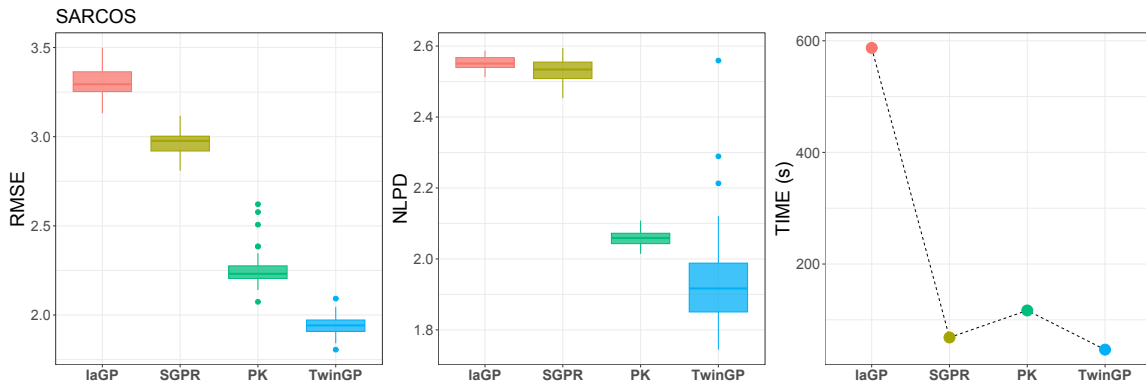


Figure 3.8: Distribution of RMSE and NLPD over 50 iterations of modeling the sarcos robotics dataset with laGP, SGPR, PK, and TwinGP. The corresponding average computation time per iteration is given in the right most plot.

Flight delays

The flight delays dataset⁶ consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. In keeping with previous studies involving this dataset, 800,000 observations are randomly selected for this study from a total of about 120 million observations. Following Park and Apley, 2018, $d = 8$ input variables are used in modeling, they are the age of the aircraft, distance that needs to be covered, airtime, departure time, arrival time, day of the week, day of the month, and month. The response is the arrival time delay. The experiment results are given in Figure 3.9. SGPR ran out of memory on our machine, and a single iteration with PK did not finish in one hour, hence, we have excluded both SGPR and PK from the plots. Here, laGP performs better than TwinGP in terms of RMSE and NLPD, while TwinGP is twice as fast.

⁶<https://community.amstat.org/jointscsg-section/dataexpo/dataexpo2009>

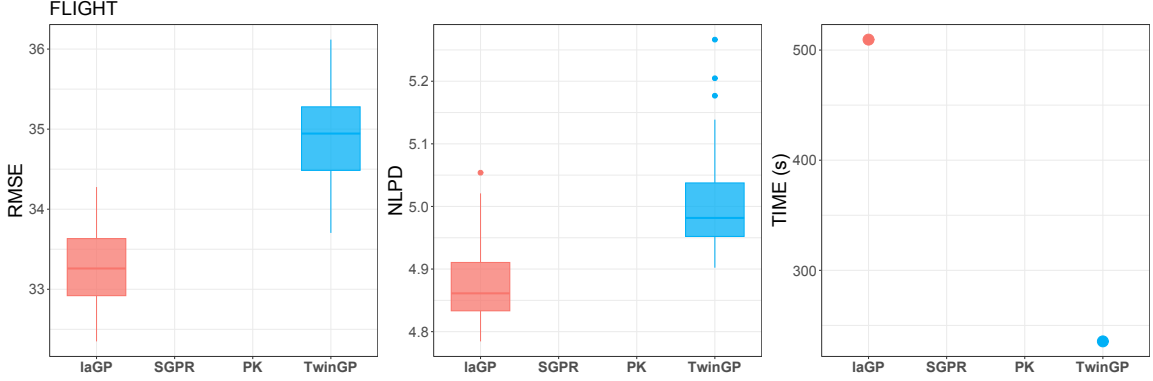


Figure 3.9: Distribution of RMSE and NLPD over 50 iterations of modeling the flight delays dataset with laGP and TwinGP. The corresponding average computation time per iteration is given in the right most plot.

3.6 Conclusions

In this article, we presented a unified global-local GP approximation that addresses the drawbacks of purely global, or local approximations in the literature. With our approximation framework, referred to as TwinGP, GP modeling on a million data points can be performed in just a few minutes on an ordinary personal computer. The two main features of TwinGP are: (i) the set of design points considered per testing location is a union of global points and local points, and (ii) the correlation function is modeled as sum of two kernels, one each to capture the global and local trend. The set of global points are identified with Twinning, and are independent of the testing location, while the local points are selected as nearest neighbors to a given testing location in the training data. The training complexity of our framework is $O(g^3)$ where g is the number of global points used, and for t testing locations, the testing complexity is $O(tg^2l)$ where l is the number of local points used. Our experiments show that the predictive performance with TwinGP is on par or better than state-of-the-art GP modeling frameworks aimed at large datasets, moreover, at a fraction of their computational cost.

Appendices

APPENDIX A

PROOFS

A.1 Proof of Proposition 1

We have that

$$\begin{aligned}
\overline{\mathbb{E}\mathbb{D}}_{n,N} &= \frac{2}{nN} \sum_{i=1}^n \sum_{j=1}^N \|\mathbf{U}_i - \mathbf{Z}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 - \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N \|\mathbf{Z}_i - \mathbf{Z}_j\|_2 \\
&= \frac{2}{nN} \sum_{i=1}^n \left\{ \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 + \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2 \right\} - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 \\
&\quad - \frac{1}{N^2} \left\{ \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 + 2 \sum_{i=1}^n \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2 + \sum_{i=1}^{N-n} \sum_{j=1}^{N-n} \|\mathbf{V}_i - \mathbf{V}_j\|_2 \right\} \\
&= \frac{2(N-n)}{nN^2} \sum_{i=1}^n \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2 - \frac{(N-n)^2}{n^2 N^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 - \frac{1}{N^2} \sum_{i=1}^{N-n} \sum_{j=1}^{N-n} \|\mathbf{V}_i - \mathbf{V}_j\|_2 \\
&= \frac{(N-n)^2}{N^2} \cdot \left\{ \frac{2}{n(N-n)} \sum_{i=1}^n \sum_{j=1}^{N-n} \|\mathbf{U}_i - \mathbf{V}_j\|_2 - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \|\mathbf{U}_i - \mathbf{U}_j\|_2 \right. \\
&\quad \left. - \frac{1}{(N-n)^2} \sum_{i=1}^{N-n} \sum_{j=1}^{N-n} \|\mathbf{V}_i - \mathbf{V}_j\|_2 \right\} \\
&= \frac{(N-n)^2}{N^2} \cdot \overline{\mathbb{E}\mathbb{D}}_{n,N-n} = (1-\gamma)^2 \cdot \overline{\mathbb{E}\mathbb{D}}_{n,N-n}.
\end{aligned}$$

△

A.2 Proof of Proposition 2

To see that the optimization in (2.6) is indeed \mathcal{NP} -hard, consider the special case where $n = N/2$, so that we get

$$\{\mathbf{U}_i^*\}_{i=1}^{N/2} = \arg \min_{\{\mathbf{U}_i\}_{i=1}^{N/2} \in \mathcal{D}} \frac{4}{N^2} \sum_{i=1}^{N/2} \sum_{j=1}^N \|\mathbf{U}_i - \mathbf{Z}_j\|_2 - \frac{4}{N^2} \sum_{i=1}^{N/2} \sum_{j=1}^{N/2} \|\mathbf{U}_i - \mathbf{U}_j\|_2$$

$$\begin{aligned}
&= \arg \min_{\{\mathbf{U}_i\}_{i=1}^{N/2} \in \mathcal{D}} \frac{4}{N^2} \sum_{i=1}^{N/2} \left(\sum_{j=1}^{N/2} \|\mathbf{U}_i - \mathbf{U}_j\|_2 + \sum_{j=1}^{N/2} \|\mathbf{U}_i - \mathbf{V}_j\|_2 \right) - \frac{4}{N^2} \sum_{i=1}^{N/2} \sum_{j=1}^{N/2} \|\mathbf{U}_i - \mathbf{U}_j\|_2 \\
&= \arg \min_{\{\mathbf{U}_i\}_{i=1}^{N/2} \in \mathcal{D}} \frac{4}{N^2} \sum_{i=1}^{N/2} \sum_{j=1}^{N/2} \|\mathbf{U}_i - \mathbf{V}_j\|_2, \tag{A.1}
\end{aligned}$$

where $\{\mathbf{V}_j\}_{j=1}^{N/2} = \mathcal{D} \setminus \{\mathbf{U}_i\}_{i=1}^{N/2}$. The optimization problem stated in (A.1) computes an edge-weighted minimum bisection of a complete graph with N nodes corresponding to the N data points in \mathcal{D} , and the Euclidean distance (ℓ_2 norm) between the nodes as the edge weights. Since graph bisection is known to be \mathcal{NP} -hard for general graphs (Garey et al., 1974), we have that the optimization in (2.6) is \mathcal{NP} -hard.

Λ

REFERENCES

- Ayres-de-Campos, D., Bernardes, J., Garrido, A., Marques-de-Sa, J., & Pereira-Leite, L. (2000). Sisporto 2.0: A program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, 9(5), 311–318.
- Ba, S., & Joseph, V. R. (2012). Composite gaussian process models for emulating expensive functions. *The Annals of Applied Statistics*, 1838–1860.
- Basak, S., Petit, S., Bect, J., & Vazquez, E. (2022). Numerical issues in maximum likelihood parameter estimation for gaussian process interpolation, In *Machine learning, optimization, and data science: 7th international conference, lod 2021, grasmere, uk, october 4–8, 2021, revised selected papers, part ii*. Springer.
- Benmeida, M. (2017). NYC taxi trip durations: Data augmentation using OSRM.
- Blanco, J. L., & Rai, P. K. (2014). Nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees.
- Bowden, G. J., Maier, H. R., & Dandy, G. C. (2002). Optimal division of data for neural network models in water resources applications. *Water Resources Research*, 38, 2-1–2-11.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.
- Brooks, T. F., Pope, D. S., & Marcolini, M. A. (1989). *Airfoil self-noise and prediction* (Vol. 1218). National Aeronautics; Space Administration, Office of Management . . .
- Chalupka, K., Williams, C. K. I., & Murray, I. (2013). A framework for evaluating approximation methods for gaussian process regression. *Journal of Machine Learning Research*, 14(1), 333–350.
- Chen, T., & Ren, J. (2009). Bagging for gaussian process regression. *Neurocomputing*, 72(7-9), 1605–1610.
- Chen, W. Y., Mackey, L., Gorham, J., Briol, F.-X., & Oates, C. (2018). Stein points (J. Dy & A. Krause, Eds.). In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning*, Stockholmsmässan, Stockholm Sweden, PMLR.
- Chen, Y., Welling, M., & Smola, A. (2010). Super-samples from kernel herding. *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 109–116.
- Dancik, G. M., & Dorman, K. S. (2008). mlegp: Statistical analysis for computer models of biological systems using R. *Bioinformatics*, 24(17), 1967.

- Das, K., & Srivastava, A. N. (2010). Block-gp: Scalable gaussian process regression for multimodal data. *2010 IEEE International Conference on Data Mining*, 791–796.
- Deisenroth, M., & Ng, J. W. (2015). Distributed gaussian processes, In *International conference on machine learning*. PMLR.
- Dette, H., & Pepelyshev, A. (2010). Generalized latin hypercube design for computer experiments. *Technometrics*, 52(4), 421–429.
- Dua, D., & Graff, C. (2017). *UCI machine learning repository*. <http://archive.ics.uci.edu/ml>.
- Elo, I., Rodriguez, G., & Lee, H. (2001). Racial and neighborhood disparities in birthweight in philadelphia. *A. Meet. Population Association of America, Washington DC*.
- Emery, X. (2009). The kriging update equations and their application to the selection of neighboring data. *Computational Geosciences*, 13(3), 269.
- Evet, I. W., & Spiehler, E. J. (1989). Rule induction in forensic science. In *Knowledge based systems* (pp. 152–160).
- Fang, K. -T., & Wang, Y. (1994). *Number-theoretic methods in statistics*. Boca Raton, FL, Chapman & Hall.
- Faraway, J. J. (2015). *Linear models with r* (Vol. 552). John Wiley & Sons.
- Fasshauer, G. E. (2007). *Meshfree approximation methods with matlab* (Vol. 6). World Scientific.
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, 179–188.
- Flury, B. (1990). Principal points. *Biometrika*, 77, 33–41.
- Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1–22.
- Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3), 209–226.
- Galvão, R. K. H., Araujo, M. C. U., José, G. E., Pontes, M. J. C., Silva, E. C., & Saldanha, T. C. B. (2005). A method for calibration and validation subset partitioning. *Talanta*, 67(4), 736–740.

- Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., & Wilson, A. G. (2018). GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration, In *Advances in neural information processing systems*.
- Garey, M. R., Johnson, D. S., & Stockmeyer, L. (1974). Some simplified np-complete problems, In *Proceedings of the sixth annual acm symposium on theory of computing*.
- Gneiting, T. (2002). Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83, 493–508.
- Gramacy, R. B. (2016). laGP: Large-scale spatial modeling via local approximate gaussian processes in R. *Journal of Statistical Software*, 72(1), 1–46.
- Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press.
- Gramacy, R. B., & Apley, D. W. (2015). Local gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, 24(2), 561–578.
- Gramacy, R. B., & Lee, H. K. H. (2008). Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483), <https://doi.org/10.1198/016214508000000689>, 1119–1130.
- Gramacy, R. B., & Lee, H. K. (2012). Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22, 713–722.
- Guha, S., Hafen, R., Rounds, J., Xia, J., Li, J., Xi, B., & Cleveland, W. S. (2012). Large complex data: Divide and recombine (d&r) with rhipe. *Stat*, 1(1), 53–67.
- Harari, O., & Steinberg, D. M. (2014). Convex combination of gaussian processes for bayesian analysis of deterministic computer experiments. *Technometrics*, 56(4), 443–454.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. New York, Springer.
- Hickernell, F. J. (1999). Goodness-of-fit statistics, discrepancies and robust designs. *Statistics and Probability Letters*, 44, 73–78.
- Joseph, V. R., Dasgupta, T., Tuo, R., & Wu, C. F. J. (2015). Sequential exploration of complex surfaces using minimum energy designs. *Technometrics*, 57(1), 64–74.
- Joseph, V. R., & Mak, S. (2021). Supervised compression of big data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 14(3), 217–229.

- Kaufman, C. G., Bingham, D., Habib, S., Heitmann, K., & Frieman, J. A. (2011). Efficient emulators of computer experiments using compactly supported correlation functions, with an application to cosmology. *The Annals of Applied Statistics*, 5(4), 2470–2492.
- Keerthi, S., & Chu, W. (2005). A matching pursuit approach to sparse gaussian process regression, In *Advances in neural information processing systems*, MIT Press.
- Kenett, R. S., & Zacks, S. (2021). *Modern industrial statistics: With applications in r, minitab, and jmp*. John Wiley & Sons.
- Kennard, R. W., & Stone, L. A. (1969). Computer aided design of experiments. *Technometrics*, 11(1), 137–148.
- Lawrence, N., Seeger, M., & Herbrich, R. (2002). Fast sparse gaussian process methods: The informative vector machine, In *Advances in neural information processing systems*, MIT Press.
- Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., & Lin, X. (2019). Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8), 1475–1488.
- Liaw, A., & Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3), 18–22.
- Mak, S., & Joseph, V. R. (2018a). Support points. *The Annals of Statistics*, 46, 2562–2592.
- Mak, S. (2019). Support points. R package version 0.1.4. <https://cran.r-project.org/src/contrib/Archive/support>.
- Mak, S., & Joseph, V. R. (2018b). Projected support points: A new method for high-dimensional data reduction. *arXiv preprint arXiv:1708.06897*.
- Mak, S., & Joseph, V. R. (2018c). Support points. *The Annals of Statistics*, 46(6A), 2562–2592.
- May, R., Maier, H., & Dandy, G. (2010). Data splitting for artificial neural networks using som-based stratified sampling. *Neural Networks*, 23(2), 283–294.
- Morris, M. D., Mitchell, T. J., & Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: Use of derivatives in surface prediction. *Technometrics*, 35(3), 243–255.
- Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., & Ford, W. B. (1994). The population biology of abalone (*haliotis* species) in tasmania. i. blacklip abalone

- (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report, 48*, p411.
- Niederreiter, H. (1992). *Random number generation and quasi-monte carlo methods*. Philadelphia, PA, SIAM.
- Owen, A. B. (2013). *Monte carlo theory, methods and examples*. <https://statweb.stanford.edu/~owen/mc/>.
- Park, C., & Apley, D. (2018). Patchwork kriging for large-scale gaussian process regression. *The Journal of Machine Learning Research, 19*(1), 269–311.
- Park, C., & Huang, J. Z. (2016). Efficient computation of gaussian process regression for large spatial data sets by patching local gaussian processes. *Journal of Machine Learning Research, 17*(174), 1–29.
- Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research, 6*(65), 1939–1959.
- Rasmussen, C., & Ghahramani, Z. (2001). Infinite mixtures of gaussian process experts, In *Advances in neural information processing systems*, MIT Press.
- Rasmussen, C. E., & Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts, The MIT Press.
- Reitermanová, Z. (2010). Data splitting. *WDS'10 Proceedings of Contributed Papers, Part I*, 31–36.
- Rodriguez, J. D., Perez, A., & Lozano, J. A. (2010). Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 32*(3), 569–575.
- Sacks, J., Welch, W. J., Mitchell, T. J., & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science, 4*(4), 409–423.
- Santner, T. J., Williams, B. J., & Notz, W. I. (2003). *The design and analysis of computer experiments*. New York, Springer-Verlag.
- Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal, 27*(3), 379–423.
- Slaney, M., & Casey, M. (2008). Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine, 25*(2), 128–131.

- Snee, R. D. (1977). Validation of regression models: Methods and examples. *Technometrics*, 19(4), 415–428.
- Snelson, E., & Ghahramani, Z. (2005). Sparse gaussian processes using pseudo-inputs, In *Advances in neural information processing systems*, MIT Press.
- Snelson, E., & Ghahramani, Z. (2007). Local and global sparse gaussian process approximations, In *Proceedings of the eleventh international conference on artificial intelligence and statistics*, San Juan, Puerto Rico, PMLR.
- Stevens, A., & Ramirez-Lopez, L. (2020). *An introduction to the prospectr package* [R package version 0.2.1]. R package version 0.2.1.
- Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of Royal Statistical Society–Series B*, 36, 111–146.
- Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis, In *Biomedical image processing and biomedical visualization*. International Society for Optics and Photonics.
- Székely, G. J., & Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8), 1249–1272.
- Theis, T. N., & Wong, H.-S. P. (2017). The end of moore’s law: A new beginning for information technology. *Computing in Science & Engineering*, 19(2), 41–50.
- Thodberg, H. H. (1993). *Ace of bayes: Application of neural networks with pruning* (tech. rep.). Citeseer.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society-Series B*, 58, 267–288.
- Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes, In *Proceedings of the twelfth international conference on artificial intelligence and statistics*, Clearwater Beach, Florida USA, PMLR.
- Tresp, V. (2000). A Bayesian Committee Machine. *Neural Computation*, 12(11), 2719–2741.
- Vakayil, A., Joseph, R., & Mak, S. (2021). *SPLit: Split a dataset for training and testing* [R package version 1.0]. R package version 1.0.
- Vanhatalo, J., & Vehtari, A. (2008). Modelling local and global phenomena with sparse gaussian processes, In *Proceedings of the twenty-fourth conference on uncertainty in artificial intelligence*, Helsinki, Finland, AUAI Press.

- Vijayakumar, S., & Schaal, S. (2000). Locally weighted projection regression: An $O(n)$ algorithm for incremental real time learning in high dimensional space, In *Proceedings of the seventeenth international conference on machine learning (icml 2000)*. Morgan Kaufmann.
- Wang, H., Yang, M., & Stufken, J. (2019). Information-based optimal subdata selection for big data linear regression. *Journal of the American Statistical Association*, 114(525), 393–405.
- Wendland, H. (2004). *Scattered data approximation*. Cambridge University Press.
- Wu, C. F. J., & Hamada, M. S. (2011). *Experiments: Planning, Analysis, and Optimization*. John Wiley & Sons.
- Xu, Y., & Goodacre, R. (2018). On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of Analysis and Testing*, 2, 249–262.
- Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12), 1797–1808.
- Zador, P. (1982). Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Transactions on Information Theory*, 28(2), 139–149.