APPENDIX A

COVER SHEET TO BE APPENDED TO ALL THESES

By law, copyright in your thesis belongs to you, the author, including all rights of publication and reproduction. Only the copyright holder may determine what others may do with the thesis.

STEP 1. What should the library do with your thesis? (Choose one)

☒ Share my thesis

You authorize the library to share your thesis with others according to the Creative Commons license selected in step 2. (See Appendix B for an explanation of the three options.)

☐ Place an optional embargo on my thesis

The library should not allow anybody except college officials to see its copy of my thesis until January 1st of the year _____. (e.g., 1 year=2018, 5 years=2022, 100 years=2117)

Under this option, you ask the library to prevent anyone else from accessing its copy of your thesis. At the end of the lockdown period, the library will distribute its copy of your thesis according to the license you choose below. (See Appendix B for an explanation of the three options.)

STEP 2. Select a license for your thesis. (Choose one)

☐ CC BY-NC-ND

☒ CC BY-NC-SA

☐ CC BY

The type of Creative Commons license you choose determines what others may and may not do with your thesis. (See Appendix B for an explanation of the three options.)

STEP 3. Sign.

Please note that, by choosing to make your thesis available (whether now or at a future date), you are waiving any protections of the Family Educational Rights and Privacy Act (FERPA) that may apply with respect to your thesis as of the date specified. FERPA generally restricts disclosure by the college of records related to your education.

Student Signature	JONATHAN CHE Student Name	2018/4/23 Date
Thesis Advisor Signature	Albert Y. Kim Thesis Advisor Name	2018/4/23 Date

Appendix B: Creative Commons Licenses

CC BY-NC-ND
Attribution-NonCommercial-NoDerivs

This license is the most restrictive, only allowing others to download your works and share them with others as long as they credit you, but they can't change them in any way or use them commercially.

- License Deed: http://creativecommons.org/licenses/by-nc-nd/3.0/
- Legal Code: http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode

---

CC BY-NC-SA
Attribution-NonCommercial-ShareAlike

This license lets others remix, tweak, and build upon your work non-commercially, as long as they credit you and license their new creations under the identical terms.

- License Deed: http://creativecommons.org/licenses/by-nc-sa/3.0/
- Legal Code: http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

---

CC BY
Attribution

This license lets others distribute, remix, tweak, and build upon your work, even commercially, as long as they credit you for the original creation. This is the most accommodating of licenses offered. Recommended for maximum dissemination and use of licensed materials.

- License Deed: http://creativecommons.org/licenses/by/3.0/
- Legal Code: http://creativecommons.org/licenses/by/3.0/legalcode

---

More information about Creative Commons licenses is here:
http://creativecommons.org/licenses/

Resampling Methods for Model Assessment and Selection with Extensions to Spatial Data

---

Jonathan Che

May 1st, 2018

Submitted to the
Department of Mathematics and Statistics
of Amherst College
in partial fulfillment of the requirements
for the degree of
Bachelor of Arts with honors

Faculty Advisor: Dr. Albert Y. Kim

Acknowledgements

The thesis in front of you would not have come together without the continued support of an amazing community of people. First, I want to thank Albert Y. Kim for advising this thesis. His contagious enthusiasm and willingness to involve me in his research process drove my work on this project and helped spark a passion for research that will follow me through graduate study and beyond. I would also like to thank Nick Horton, my academic advisor, for his seemingly unlimited energy and wisdom. I would not have become the student I am today without his support and advice over the course of my college career.

My gratitude extends to all of the faculty in the Department of Mathematics and Statistics. Your passion for teaching is inspiring, and I have learned so much from you all. I am particularly grateful for the Statistics and Data Science Fellows program, made possible by faculty engagement and support from David and Jeanette Rosenblum ('92), for opening my eyes to the world of possibilities within statistics.

I would like to show my appreciation for my friends, many of whom had theses of their own to write this year, for their encouragement and support throughout this thesis-writing process. Special thanks to Sarah Teichman and Tim Lee for working with me in our statistics courses and for their shared passion for data visualization, from complex interactive graphics to the humble scatterplot. I would also like to thank Emily Isko for her care and understanding through all of the ups and downs of this thesis.

Finally, I want to thank my family for being there for me every step of the way. You have always believed in me, and it is certainly your love and guidance that has pushed me to come this far. Thank you.

Contents

List of Figures

Abstract

The rapidly growing volume of data in the modern world has given researchers unprecedented opportunities to develop predictive models for a huge range of different fields. Once a model has been produced, however, some important questions arise: How well can we expect the model to perform? Is the model better than other possible models for the data? Cross-validation and the bootstrap are two classes of nonparametric methods commonly used to answer such questions. In this thesis, we begin by surveying and comparing cross-validation and bootstrap methods for assessing model performance. We then study extensions of cross-validation and the bootstrap to spatial settings, where independence assumptions break down. Next, we apply all of these methods to the problem of selecting the best model. Finally, we conduct simulation studies to compare how well the methods assess and select models in spatial contexts. Though there is more work to be done, our initial results indicate that spatial methods indeed outperform nonspatial ones when applied to model assessment and selection in spatial contexts.

Chapter 1

Introduction

The past few years have seen an explosion of interest in "machine learning": the production of models from data, broadly speaking. As the modern world continues to churn out data at unprecedented rates, researchers and practitioners alike are scrambling to develop models to make sense of all of it. Producing these models, however, is only half the battle; once produced, models need to be tested to see how well they can be expected to perform relative to other models. It has therefore become increasingly important to understand not only how to model data, but also how to assess and compare different statistical models to choose the best one for the problem at hand.

Statisticians have developed many methods to deal with this problem of *model selection*: how to choose the best model for the real-world mechanism encoded in the data. One popular and intuitive method for doing so draws on tools from *model assessment*. The idea is to first estimate how the models in consideration would perform on new data, and to then choose the model estimated to have the best performance.

Though model assessment and selection techniques can be applied to nearly any kind of data, their applications to spatial data result in some unique challenges. This is due to *spatial autocorrelation*, or, as famously stated by geographer Waldo Tobler, the first law of geography: "Everything is related to everything else, but near things are more related than distant things" [57]. The strong relationships between nearby observations in spatial data violate many of the independence assumptions underpinning popular model assessment and selection methods. To address the issues that arise, researchers have proposed a variety of spatial model assessment and selection methods.

This thesis has two primary goals. The first is to survey popular nonparametric resampling-based methods for model assessment and selection, in both nonspatial and spatial contexts. We introduce different methods and explore their statistical properties from both theoretical and simulation-based perspectives. The second goal is to see how different spatial model assessment and selection methods perform in simulations, as well as on a real-world forest ecology dataset collected by Albert Y. Kim[1] and David Allen[2].

The chapters are organized as follows. Chapter 2 introduces cross-validation and the bootstrap for model assessment and also explores their properties in terms of what they estimate, their bias, and their variance. Chapter 3 extends those methods to spatial settings and studies their properties there. Chapter 4 applies the model assessment methods from Chapters 2 and 3 to the problem of model selection in both spatial and nonspatial settings. Finally, Chapter 5 runs the abovementioned simulations to study how well spatial resampling methods perform at model assessment and selection, on both real and simulated data.

[1]Lecturer of Statistics, Amherst College

[2]Assistant Professor in Biology, Middlebury College

Chapter 2

Resampling Methods for Model Assessment

Model assessment aims to estimate how well a given model will perform on new data. Before we motivate the problem of model assessment and explore methods for tackling it, we will first develop some general background and notation for statistical modeling.[1]

2.1 Background

Broadly speaking, this thesis focuses on methods for assessing *supervised learning* models. In supervised learning, models are fit, or *trained*, on labeled *training sets* $\mathcal{T}$ of N observations $(\mathbf{x_i}, y_i)$, $i = 1, 2, \ldots, N$. Each $\mathbf{x_i}$ is a vector of *predictor variables*, and each y_i is a single *label* (or *response variable*). Predictors can generally be any kind of variable, though they are typically numeric or categorical. Labels are either numeric or categorical: supervised learning problems with numeric labels are known as *regression* problems, and problems with categorical labels are *classification* problems.

In theory, each label y_i is of the form $f(\mathbf{x_i}) + \epsilon_i$, where $f\colon \mathbf{x_i} \to y_i$ is some unknown *data-generating function* (or *signal*) of the labels from the predictors and ϵ_i is the random *irreducible error* (or *noise*) associated with each observation. The goal of supervised learning is to use the given training set $\mathcal{T}$ to construct a supervised learning model $\hat{f}\colon \mathbf{x} \to y$ that can input any $\mathbf{x}$ and output a corresponding prediction $\hat{y} = \hat{f}(\mathbf{x})$ of its label. Example 2.1.1 gives an example of a supervised learning problem, using the notation developed above.

[1]The notation and definitions in this chapter generally follow from [23], unless otherwise noted.

Example 2.1.1. An ecologist wishes to model the annual growth of trees. They collect a training set $\mathcal{T}$ of $N = 500$ trees from Forest A that records the species, age, location, and previous year's growth of each tree (in inches). So, $\mathbf{x_i} = \langle \texttt{species}_i, \texttt{age}_i, \texttt{location}_i \rangle$ and $y_i = \texttt{prev\_growth}_i$ for each tree i. Each y_i is numeric, so this is a regression problem. The ecologist might use $\mathcal{T}$ to fit a multiple regression model $\hat{f}$ that could input a previously unseen tree's species, location, and age, and output an estimate of its annual growth.

Importantly, the new inputs $\mathbf{x}$ do not have to be identical to any of the $\mathbf{x_i}$ in $\mathcal{T}$.[2] In other words, the model can predict the labels of observations that were not in the original training set. Hopefully, the predicted label $\hat{f}(\mathbf{x})$ for a new observation $\mathbf{x}$ is similar to $f(\mathbf{x})$, the true label for the observation. The goal of model assessment is to estimate how close $\hat{f}(\mathbf{x})$ is expected to be to $f(\mathbf{x})$ for a random $\mathbf{x}$ drawn from the population.

The most straightforward way to assess the performance[3] of a model is its *training error*:

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(\mathbf{x_i})).$$

Here, $L(y_i, \hat{y}_i)$ is a *loss function*, some nonnegative measurement of error, i.e., how much the predicted $\hat{y}_i = \hat{f}(\mathbf{x_i})$ differs from the actual y_i.[4] Training error measures how well a model can predict the labels of the data it is trained on. Figure 2.1 shows how the model is both trained and evaluated using the same set of data.

Figure 2.1: Diagram of training error model assessment

Typically, though, users are not interested in how well their models perform on existing training data; rather, they want to know how well their models would predict the labels

[2]Though $\mathbf{x}$ must consist of the same kinds of variables as the $\mathbf{x_i}$, e.g., if each $\mathbf{x_i}$ is a vector $\langle a, b, c \rangle$ where a and b are numeric and c is a categorical that takes on the values `red` or `blue`, then $\mathbf{x}$ must also be a vector of two numerics and either `red` or `blue`.

[3]In this thesis, model "performance" is broadly used as an inverse of out-of-sample model error, i.e., "good model performance" means "low out-of-sample model error."

[4]E.g., one common loss function for regression problems is squared error, $(y_i - \hat{y}_i)^2$.

of unseen, unlabeled observations. For example, because the ecologist from Example 2.1.1 already knows the annual growths of trees in Forest A from last year, they would likely be more interested in trying to predict their growth this year. In statistical terms, users are interested in the *out-of-sample error* (or *generalization error*) of their models.

Training error, however, can often be a very poor estimate of out-of-sample error because of *overfitting*. Intuitively, overfitting occurs when models "memorize" training data instead of extracting general rules. Recall that labels $y_i = f(\mathbf{x_i}) + \epsilon_i$ are combinations of signal and random noise. For a given training set $\mathcal{T}$, both the signal f and the random noise vector $\epsilon_\mathcal{T}$ are fixed. Ideally, a model $\hat{f_\mathcal{T}}$ trained on $\mathcal{T}$ would be similar to f. A model $\hat{f_\mathcal{T}}'$ that is overfit to $\mathcal{T}$, however, would be similar to $f + \epsilon_\mathcal{T}$. Now suppose a new dataset $\mathcal{V}$ is drawn from the same population as $\mathcal{T}$, so $\mathcal{V}$ has the same signal f but a different fixed noise vector $\epsilon_\mathcal{V}$. For $\hat{f_\mathcal{T}}$, training error would be a good estimate of out-of-sample error because the model's errors on $\mathcal{T}$ and $\mathcal{V}$ would primarily depend on $\epsilon_\mathcal{T}$ and $\epsilon_\mathcal{V}$, which are random but identically distributed (by assumption). For $\hat{f_\mathcal{T}}'$, though, training error would underestimate out-of-sample error. Because $\hat{f_\mathcal{T}}'$ accounts for $\epsilon_\mathcal{T}$, it should have a lower training error than $\hat{f_\mathcal{T}}$. $\hat{f_\mathcal{T}}'$, however, would have much higher error on $\mathcal{V}$. $\hat{f_\mathcal{T}}'$ is similar to $f + \epsilon_\mathcal{T}$, i.e., for each value of $\mathbf{x_i}$ in $\mathcal{T}$, the model learns not only $f(\mathbf{x_i})$ but also its corresponding $\epsilon_{\mathcal{T}_i}$. Though $\epsilon_\mathcal{T}$ and $\epsilon_\mathcal{V}$ are identically distributed, there is no reason to suspect that, given some $\mathbf{x_i}$ in $\mathcal{T}$, a similar value $\mathbf{y_j}$ in $\mathcal{V}$ would have $\epsilon_{\mathcal{V}_j}$ similar to $\epsilon_{\mathcal{T}_i}$. As such, the training error of $\hat{f_\mathcal{T}}'$ would dramatically underestimate its out-of-sample error on $\mathcal{V}$.

In general, supervised learning algorithms aim to produce models that fit the training data well. Though many algorithms utilize regularization methods[5] to reduce overfitting, models still tend to perform better on their training sets than on new observations, which may involve new values of $\mathbf{x_i}$ or slightly different data-generating functions. Because of this, we say that training error is an *optimistic* estimate of out-of-sample error.[6]

There are many other model assessment methods that can produce better estimates of

[5]Regularization methods (see page 168 in [23]) penalize both training error and model complexity, so they tend to produce less complex models that are less prone to overfitting.

[6]This is a colloquial use of the term "optimistic," which is similar to but not the same as the more formal term "optimism" as described in Section 7.4 of [23].

out-of-sample error. The simplest such method is the *validation set* approach. Instead of training the model on all of the given data, the data are split into two sets: a training set and a validation set. The model is then fit on the training set and used to predict the observations in the validation set. Comparing these predictions to the actual labels of the observations in the validation set provides an estimate of the model's out-of-sample error.

Figure 2.2 diagrams the validation-set approach. Unlike Figure 2.1, we see that using a validation set means that the model is not trained on any of the data used to evaluate it. The validation-set approach assumes independence between training sets $\mathcal{T}$ and validation

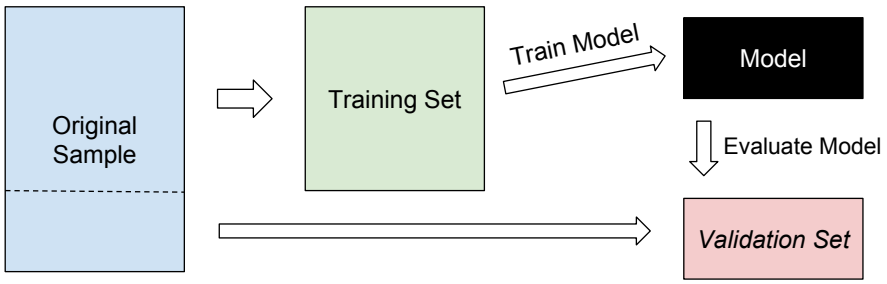


Figure 2.2: Diagram of validation-set-error model assessment

sets $\mathcal{V}$. While the observations in $\mathcal{T}$ and $\mathcal{V}$ should be drawn from the same population, the occurrence of observations in $\mathcal{T}$ should not affect the occurrence of observations in $\mathcal{V}$. If this assumption holds, then $\mathcal{V}$ represents an independent, out-of-sample set of new data, so the model's error on $\mathcal{V}$ should be similar to its out-of-sample error. Importantly, while overfitting to $\mathcal{T}$ by modeling $f + \epsilon_{\mathcal{T}}$ improves training error, it typically hurts validation error on $\mathcal{V}$, which is generated by $f + \epsilon_{\mathcal{V}}$.

While the validation-set approach is effective at model assessment, it has a major drawback. In many situations, it is too costly to sacrifice training data to validate models. Setting aside data for validation means reducing the size of the sample used for training, which can lead to weaker predictive models and less certainty about the significances of the relationships in the data. To address this issue, techniques have been developed to estimate the out-of-sample error rates of models using all of the given data.

In this chapter, we will review two such methods that can be used to estimate a model's out-of-sample error: cross-validation and the bootstrap.

2.2 Cross-Validation

Cross-validation (CV) is a popular model-assessment method. Though researchers had begun to notice the overoptimism of using training error to estimate model performance as early as 1931 (see [36]), CV's high computational cost meant that it would not become practical until the advent of cheap computing later in the century. Some researchers began studying CV again in the late 1960's, and a comprehensive 1974 survey of CV techniques in [54] is commonly pointed to as the key review of early research on the method [3].

Like the validation-set approach, CV estimates out-of-sample error by splitting the given data into training and validation sets. Rather than doing so only once, however, CV splits it multiple times and averages errors over all splits. In this manner, CV can use all of the given data to train models while still evaluating the models on observations that are effectively out-of-sample. It should be noted that for accurate model assessment, the same independence conditions as for the validation-set approach apply; the observations in each resampled validation set should resemble a new sample from the population.

K-fold Cross-Validation

The most popular CV method is *K-fold cross-validation* (Algorithm 1). K-fold CV resamples the given data by first randomly partitioning it into K roughly equally sized folds. Each fold is treated as the validation set in one of the K iterations of the algorithm. For example, Figure 2.3 shows the how the given data is split into training and validation sets for 3-fold CV. The final output of K-fold CV is the average error over the K validation sets.

Algorithm 1 K-fold Cross-Validation

1: Randomly partition data $(\mathbf{x_i}, y_i)$ into K equally sized folds F_k
2: **for** each fold F_k **do**
3: `train` $\leftarrow \{(\mathbf{x_i}, y_i) \mid (\mathbf{x_i}, y_i) \notin F_k\}$
4: `validate` $\leftarrow \{(\mathbf{x_i}, y_i) \mid (\mathbf{x_i}, y_i) \in F_k\}$
5: Train model m_k on `train`
6: Record performance p_k of m_k on `validate`
7: **return** $\frac{1}{K}\sum_{k=1}^{K} p_k$

There are many other methods that can be used to randomly split data for CV.[7] Though

[7] A more detailed survey of different CV methods can be found in [3].

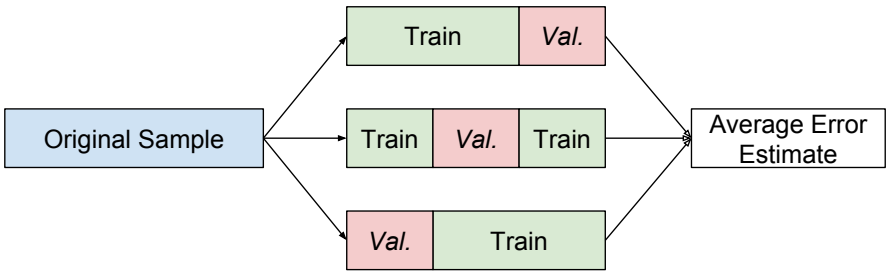


Figure 2.3: Example of 3-fold CV

this thesis will touch on some of them, it will primarily focus on K-fold CV and the related *leave-one-out cross-validation* (LOOCV), which is K-fold CV when $K = N$, the number of observations in the data.

Repeated K-fold Cross-Validation

Modern applications of K-fold CV rarely run CV only once, since CV estimates of error are known to have high variance.[8] In *repeated K-fold cross-validation*, the K-fold CV algorithm is run multiple times, with different random partitions of the data in each iteration. The average error over all of the runs is used for a less variable estimate of the prediction error.

2.3 Bootstrap

Another method that can be used to estimate out-of-sample error is the (nonparametric[9]) *bootstrap*, originally introduced by Efron in [17]. The bootstrap is a resampling method typically used to estimate the means and variances of population parameters from empirical samples of data. Notably, the bootstrap is based on resampling with replacement, unlike CV, which resamples without replacement. The algorithm is shown in Algorithm 2, where B is a user-defined number of bootstrap samples to generate and N is the size of the original dataset. In the context of model assessment, s_i is some measure of the error of the model trained on `sample` when evaluated on the original training data.

[8] Details in Section 2.5.

[9] Since the focus of this thesis is on nonparametric methods, we choose not to cover the parametric bootstrap, even though it is a useful method.

Algorithm 2 The Bootstrap

1: **for** i in $1 \dots B$ **do**
2: `sample` $\leftarrow$ Random selection of N observations, with replacement, from data
3: Compute and record statistic s_i on `sample`
4: **return** $\bar{s} = \frac{1}{B}\sum_{i=1}^{B} s_i$ and $v = \frac{1}{B-1}\sum_{i=1}^{B}(s_i - \bar{s})^2$

The theoretical foundation of the bootstrap depends on similar assumptions as CV. To borrow a phrase from [34], "the 'bootstrap world' should closely approximate the real world." The bootstrap assumes that the empirical distribution $\hat{F}$ of the given sample of data is a good estimate of the true distribution F of the population from which it is drawn, i.e., that the given sample is representative of the population. If this assumption is held,[10] the B bootstrap samples mimic B new random samples from the population, which means that the empirical distribution of s_i should be a good approximation of the sampling distribution of the true s for random samples taken from the population.

Leave-One-Out Bootstrap

A *naive bootstrap* estimate of model error would be to let s_i be the error of the model trained on `sample` when evaluated on all of the original data. Clearly, using the naive bootstrap estimate leads to the same issues as does using training error; the model is trained and tested on overlapping data, which awards overfitting. The *leave-one-out bootstrap* (LOO bootstrap) addresses this issue [23]. In the LOO bootstrap, the error for each point is averaged only over models trained on bootstrap datasets that do not contain the point.[11] The final error estimate is the average error over all points in the training set.

.632 and .632$^+$ Bootstrap

Extensions to the bootstrap for model assessment, the .632 and .632$^+$ bootstraps, are introduced in [18] and [20]. The .632 bootstrap estimate of out-of-sample error is

$$\hat{Err}^{(.632)} = .368 * \overline{err} + .632 * \hat{Err}^{(1)},$$

[10] In general, this assumption cannot be checked. Note, though, that $\hat{F}$ is the maximum likelihood estimate of F, i.e., it is the distribution that has the highest chance of producing the given sample of data [19].

[11] Note that since bootstrap datasets are randomly chosen, this means that each point's error may be averaged over a different number of model predictions.

where $\overline{err}$ is the training error and $\hat{Err}^{(1)}$ is the LOO bootstrap error estimate. The $.632^+$ estimate is similar, with the .368 and .632 weights themselves weighted based on the relative overfitting of the model compared to random guessing. More overfit models have errors estimated by a method more similar to the LOO bootstrap, less overfit models the .632 bootstrap. The intuitions behind these modifications of the bootstrap have to do with the bias-variance tradeoff, and we delay their discussion until Section 2.5.2.

2.4 Conditional vs. Expected Test Error

The various model-assessment methods described in this chapter lead to a natural question: given some data and a model, what method should be used to assess the model? Alas, there is no single best model-assessment method to use in all situations. Luckily, these methods can be compared in terms of the well-known *bias-variance tradeoff* in statistics.

Before we can discuss the bias-variance tradeoff, however, we need to answer the question: the bias and variance of the estimate of *what*? Up to this point, we have seen how CV and the bootstrap estimate out-of-sample error, generally speaking. Clearly distinguishing the specific type of error that these methods estimate, though, reveals some important theoretical and practical complications.

Given a training set $\mathcal{T} = (\mathbf{x_i}, y_i)$, there are two kinds of out-of-sample error that one might want to estimate. The first is *conditional test error*:

$$Err_{\mathcal{T}} = \mathbb{E}_{\mathbf{x_i}, y_i}[L(y_i, \hat{f}(\mathbf{x_i}))|\mathcal{T}].$$

Conditional test error is the error that a model trained on $\mathcal{T}$ is expected to have on points drawn from the same population as $\mathcal{T}$. The second kind of error is *expected test error*:

$$Err = \mathbb{E}_{\mathcal{T}}[\mathbb{E}_{\mathbf{x_i}, y_i}[L(y_i, \hat{f}(\mathbf{x_i}))|\mathcal{T}]],$$

which is the expected error of a model trained on a random training set of the same size as $\mathcal{T}$ drawn from the population [23].

Though these two errors look similar, they have distinct implications in different contexts. In many supervised-learning applications, practitioners are given some data $\mathcal{T}$ to train a model that will have the best possible performance on new data. Models in these

situations should be assessed based on their conditional errors, since the goal is to accurately estimate the predictive performance of the model trained on $\mathcal{T}$. In other applications, though, researchers are less interested in assessing the performance of a model trained on a particular $\mathcal{T}$. For example, if a researcher wants to know which one of two models better describes relationships in the general population, they might want to know how each model would perform on an arbitrary training set drawn from the population. We will see more examples of this latter objective in Chapter 4, which discusses model selection.

So which type of test error do CV and the bootstrap estimate? For model assessment, we want to estimate conditional error. One might suspect that a method like LOOCV, which fits models using nearly all of the original training data, would provide a good estimate of conditional error. Yet because K-fold CV and the bootstrap partition the original data into training and validation sets, the methods only ever train models on training sets that are sampled from the original in different ways. Thus, one might also suspect that CV and the bootstrap should provide better estimates of expected test error [23].

Simulation studies support the latter conclusion. For K-fold CV, simulation studies show that K-fold CV estimates of error are actually *negatively* correlated with conditional error (see Section 12.2 of [19], Section 7.12 of [23]). [23] concludes that "estimation of test error for a particular training set is not easy in general, given just the data from that same training set. Instead, cross-validation and related methods may provide reasonable estimates of the *expected* error Err."[12] To see if similar results hold for the bootstrap, LOO bootstrap, and .632 bootstrap, we replicate the simulation in [23] using both K-fold CV and the bootstrap methods.

Details of our replication can be found in Appendix A, but the main results are shown in Figures 2.4 and 2.5. We simulate datasets and fit best subset linear regression models for different values of p, the number of selected predictors. For each model, we use the simulated datasets to estimate error using the naive bootstrap, LOO bootstrap, .632 bootstrap, and 10-fold CV. We also compute the "true" $Err_{\mathcal{T}}$ and Err for each model and dataset.

Figure 2.4 plots the absolute value of the average difference between $Err_{\mathcal{T}}$ & Err and

[12] A brief survey of some other literature on whether K-fold CV estimates conditional or expected error can be found in [5], which also concludes that K-fold CV serves as a better estimator of expected error.

the error estimates. We see that the dotted lines generally lie below their corresponding solid lines, i.e., that all of the error estimation methods generally produce estimates that are closer to Err (the dotted lines) than $Err_{\mathcal{T}}$ (the solid lines).

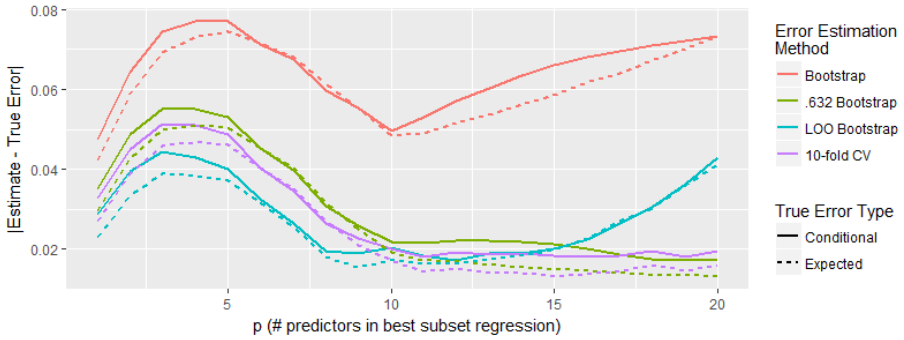


Figure 2.4: How well error-estimation methods estimate $Err_{\mathcal{T}}$ and Err

Figure 2.5 plots the correlations between the error estimates and $Err_{\mathcal{T}}$. While we see a consistent nonzero correlation for most values of p, we notice that this correlation is indeed negative. While these results are discouraging, nonparametric methods like CV and the bootstrap are still the most intuitive methods to estimate test error for many models. As such, it remains important to study their properties; indeed, one could argue that the results of this section make it all the more important to precisely study how they behave.

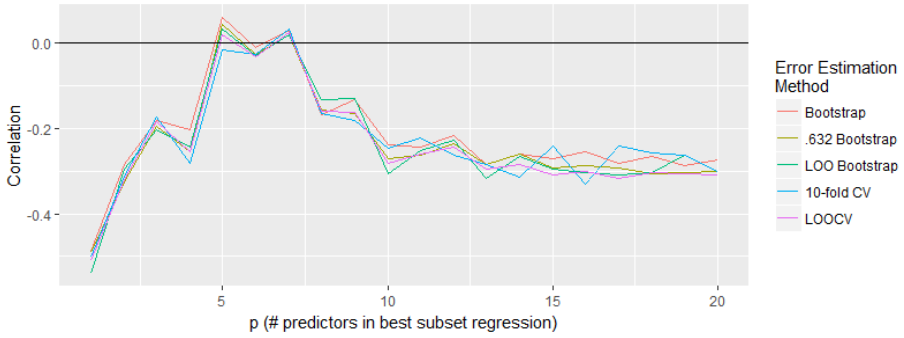


Figure 2.5: Correlation between error estimates and $Err_{\mathcal{T}}$

2.5 The Bias-Variance Tradeoff in Model Assessment

Now we return to the discussion of the bias-variance tradeoff. In the context of model assessment, the tradeoff is between the bias and variance of the error estimates produced

by CV and the bootstrap.[13] This section provides some intuition for this tradeoff.

Most models perform better when trained on more data. *Learning curves* visualize this improvement by plotting the validation accuracy of a model against the size N_t of its training set. Figure 2.6 shows how models typically have steep gains in accuracy as small training sets grow larger (to the left of the red line), with smaller gains as $N_t \to \infty$ [23].

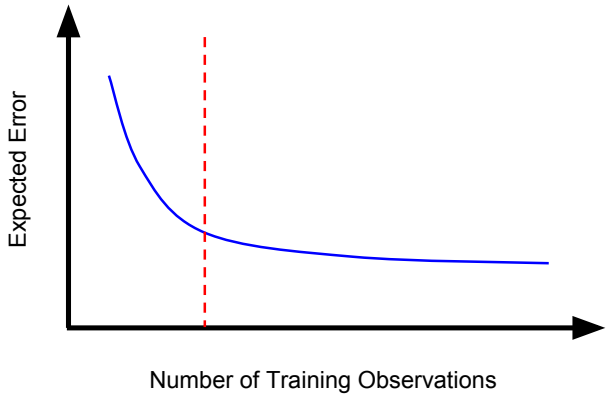


Figure 2.6: Hypothetical model learning curve

The learning curve tells us that, all things remaining equal, a model trained on all of the given data will tend to perform better on out-of-sample data than a model trained on only some of the data, particularly if the original dataset is small. In holding out data to use for validation, then, CV and the bootstrap pessimistically bias (i.e., inflate) the error-rate estimates for the model; the trained models are trained on only some of the given data, so they will generally perform more poorly on the validation sets than would models trained on all of the given data and evaluated on new data. The more data held out for validation, the more biased the error estimate will be.

Holding out less data for validation, though, increases the variance of the out-of-sample error estimates. When very few observations are held out for validation in a resampling method, the training datasets are all very similar to the original full dataset. So, the trained models will all be very similar to both each other and the model trained on all of the data. In other words, the trained models will all be highly dependent on the particular training

[13] Note that this is not the bias and variance of the model $\hat{f}$ itself, which would consider how, given some $\mathbf{x}$, $\hat{f}(\mathbf{x})$ differs from $f(\mathbf{x})$ across different $\mathcal{T}$ drawn from the population.

set originally sampled from the population. If a new training set were drawn from the population, all of the trained models would change significantly (while still being similar to each other). This between-model dependence leads to large changes in out-of-sample error estimates for different draws from the population, i.e., high variance [23].[14]

In summary, as resampling methods for model assessment hold out more data in each iteration, they produce less variable but more biased estimates of out-of-sample error. There is an extensive literature comparing the bias and variance of different CV and bootstrap methods for model assessment, which we will briefly review in the following sections.

Computation Time Tradeoff

Computation time also gets "traded." For K-fold CV, the cost of decreasing bias is increased computation time, since more folds need to be computed. Obviously, repeating K-fold CV requires more computation time as well. For the bootstrap, increasing computation time (by increasing B) produces better estimates of the bias and variance of the error estimate. Computation time was a significant concern in the 1980's and 1990's, when much of the original literature on CV and the bootstrap was being developed. Though it is definitely still very important to consider in many modern applications, this thesis will not focus on computation time when comparing model assessment methods.

2.5.1 Bias-Variance in K-fold Cross-Validation

In K-fold CV, bias and variance are controlled by the user-defined parameter K. Figure 2.7 gives an overview of how K affects bias and variance. In general, we will see that as K increases (from 2 to N), bias decreases but variance increases.[15]

The behavior of the bias of K-fold CV is well-documented. [3] shows that for independent training and validation sets, bias depends on the difference in performance of the algorithm trained on N_t versus N observations, where N_t is the number of training observations in

[14] Alternatively, one could think of the error estimates for each of the R resamples r as random variables E_r, so the overall error estimate would be $\frac{1}{R}\sum_{r=1}^{R} E_r$. Since $Var(\sum_{r=1}^{R} E_r) = \sum_{r=1}^{R}\sum_{s=1}^{R} Cov(E_r, E_s)$, we see that the variance of the overall estimate increases as the covariance between the errors estimated based on different resamples increases.

[15] The one notable exception is in the cases of very low K (e.g., $K = 2$ or 3). In these cases, variance is higher due to *partition sensitivity*, which will be discussed later in this section.

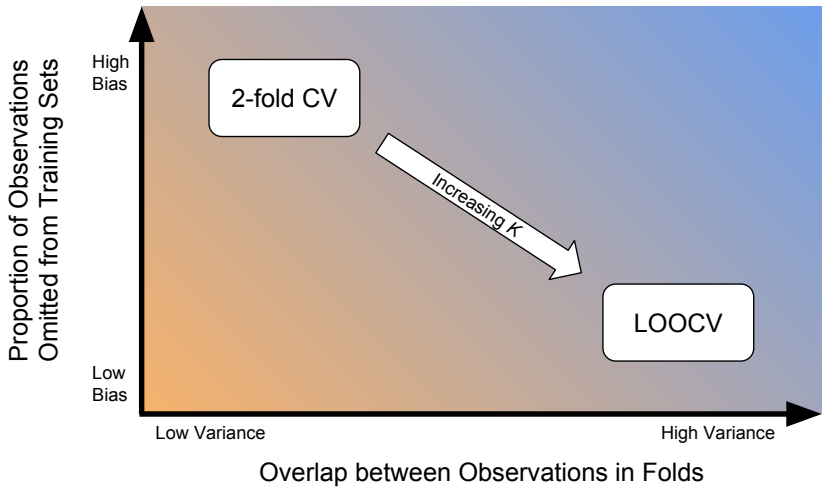


Figure 2.7: Bias-variance tradeoff in K-fold CV

a given fold.[16] Since models improve as they are trained on more observations, K-fold CV is negatively biased, but less so as N_t approaches N. Recalling Figure 2.6, we note that choosing values of K that result in training sets that are too small (i.e., to the left of the red line) would likely result in steep increases in bias.

The behavior of the variance of K-fold CV is a much more challenging problem in general. Ideally, one would compute the variance of K-fold CV for a given dataset with N observations by drawing many independent datasets of N observations from the same population and simply computing the sample variance of the K-fold CV error estimates on those new datasets. Of course, we typically use CV only when it is difficult to draw new observations from the population, so the variance of the CV estimator needs to be estimated using only the given dataset. Unfortunately, [5] proves that there are no unbiased estimators

[16] Another interesting intuition for the bias of K-fold CV can be found in [34]. [34] considers the extreme example of a classification model that makes the same predictions for each observation i in the full training set $\mathcal{T}$, regardless of which of the K training sets $\mathcal{T}_K$ produced by K-fold CV it is trained on. Such a model is said to be *stable*. In this case, the classification accuracy follows a binomial(n, p) distribution, where each of the n observations in $\mathcal{T}$ has a probability p of being classified correctly, where p is the true accuracy of the classifier. Thus, in this case, the CV estimate of error would be unbiased. Of course, most models are not perfectly stable. Holding out fewer observations, though, would lead to more consistent predictions across the folds, i.e., more stability. Intuitively, increasing the model stability in this way would approach the extreme situation considered by Kohavi, where the CV estimate of error is unbiased. Thus, holding out fewer observations in CV should lead to a less biased error estimate.

of the variance of the K-fold CV estimate of the expected test error. Without getting into the details of their proof, this is because the overlaps between the training sets used for K-fold CV produce dependencies between error estimates, so simply using the population variance of the K error estimates underestimates the true variance.

Regardless, [3] provides some basic intuition for the variance of CV methods. It shows that the variance of a validation-set estimate of model error $\widehat{Err}_{\mathcal{V}}$ is roughly proportional to $\frac{1}{N_v}$ plus the variability of the model trained on N_t observations, where N_v is the size of the validation set and N_t is the size of the corresponding training set. The formula shows that as N_v increases, the variance of $\widehat{Err}_{\mathcal{V}}$ decreases. Also, it shows that the variance of $\widehat{Err}_{\mathcal{V}}$ depends on the stability of the given model, i.e., on how much the model's outputs vary based on its particular training set. [44] points out that most models become more stable as their training sets get bigger, so the variance of $\widehat{Err}_{\mathcal{V}}$ should decrease as N_t increases. While the intuition described in this paragraph is useful, it becomes complicated when applied to K-fold CV, since N_t and N_v are inversely related and depend directly on N and K. Also, model stability varies dramatically across models and sample sizes. Simulation studies are therefore quite useful for better understanding the variance of K-fold CV.

In general, simulation studies (see Section 2.5.3) show that K-fold CV error estimates can be highly variable. [49] decomposes this variance into *training sensitivity* (TS) and *partition sensitivity* (PS). Given a fixed number of folds, TS comes from how the CV training sets differ from the original training set; PS comes from the random ways in which points can be assigned into the training/validation folds. [49] shows that TS is generally much greater than PS, and that PS is large for small K but approaches 0 as $K \to N$.[17] This behavior of PS is due to the fact that there are simply more ways to assign points into K folds for low values of K than for higher values of K. For example, there are $\binom{N}{N/2}$ ways to partition N observations into two folds, which maximizes $\binom{N}{k}$ for all k. The high value of PS for small K explains the phenomenon (mentioned in Footnote 15) where K-fold CV is more variable for small K: while TS typically dominates PS, PS is large enough for small K that it contributes a significant amount of variability to the K-fold CV error estimate.

[17] For LOOCV, PS=0, since there is only one way in which N points can be assigned into N folds.

[49] considers the use of repeated K-fold CV to reduce the variance of the single K-fold CV estimate. It shows that repeated K-fold CV decreases PS and produces a stronger positive relationship between K and variance. These results make intuitive sense: PS is large for small values of K because of the large number of possible partitions of the data into training and validation sets, and repeating the K-fold CV procedure accounts for more of these possible partitions in the error estimate.

[58], however, cautions against blindly using repeated CV to produce more precise estimates of model error. It points out that because CV does not actually $Err_{\mathcal{T}}$, using repeated CV to increase the estimate's precision can be counterproductive. Often, standard-error estimates are used to construct confidence intervals. Reducing standard errors reduces the sizes of those intervals. Thus, repeated CV can actually increase the chance that $Err_{\mathcal{T}}$ lies outside of the estimated confidence interval, because CV does not estimate $Err_{\mathcal{T}}$ well.

[62] provides a more theoretical argument against repeated CV. CV produces a point estimate $\overline{p} = \frac{1}{K}\sum_{k=1}^{K} p_k$ of model error. Repeated CV reduces the variance of $\overline{p}$ by essentially taking more samples of p_k. [62], though, notes that the variance of the sum only decreases if the p_k are independent. While the p_k within the same round of CV are independent, the p_k from across different rounds of CV are not, since they are produced using the same data. Because the p_k are positively correlated with each other across rounds, the variance estimate of $\overline{p}$ produced by repeated CV is artificially low (because it does not account for this correlation). Regardless, many modern practitioners still utilize repeated CV rather than single CV, and we will consider both methods moving forward.

2.5.2 Bias-Variance in the Bootstrap

Unlike K-fold CV, the bias and variance of the bootstrap cannot be easily tuned by a single parameter (i.e., K). Instead, this section will discuss the bias and variance of bootstrap estimates produced by the LOO bootstrap, .632 bootstrap, and .632+ bootstrap.

The LOO bootstrap has bias similar to that of 3-fold CV, because given a dataset of size N, the probability that observation i will be in a bootstrap sample of the dataset is:

$$1 - \mathbb{P}(i \text{ not chosen in each of } N \text{ draws}) = 1 - (1 - \frac{1}{N})^N \xrightarrow{N\to\infty} 1 - \frac{1}{e} \approx .632.$$

Because .632 of the original observations are expected to be used for training in each LOO bootstrap sample, the pessimistic bias of the LOO bootstrap is similar to the bias of 3-fold CV, which uses two-thirds of the original observations in each resample.

The .632 bootstrap accounts for this bias by using a weighted average of the pessimistic LOO bootstrap error estimate and the optimistic training error estimate. Though the theoretical justification behind the weighted average is weak (see [18]), Efron justifies the method roughly as follows. In general, models perform worse on points that are different from the points in their training sets. For example, training error $\overline{err}$ underestimates true error because the points it uses to estimate error are identical to its training points. The LOO bootstrap has the opposite problem. Because it only evaluates models on training-set points *not* in the bootstrapped training set, the points it uses for evaluation actually differ more, on average, from its training set than would independently drawn evaluation points. Thus, the LOO bootstrap's pessimism can be attributed to the fact that the points that it uses to estimate model error are, on average, less similar to its training sets than would be expected of a random draw from the population. Efron shows that the points that contribute to LOO bootstrap error are, on average, $\frac{1}{.632}$ "farther" from the bootstrapped training set than an independent test point would be expected to be from the original training set. Thus, using the weighted average $\hat{Err}^{(.632)} = .368 * \overline{err} + .632 * \hat{Err}^{(1)}$ is roughly equivalent to considering error on points "about as far" as independent test points would be expected to be. The .632 bootstrap should therefore be less biased than either the training error, which considers points "too close" to the training set, or the LOO bootstrap error, which considers points "too far" from the training set.

The .632 bootstrap is still biased in some pathological cases, e.g., it is too optimistic in cases of overfitting where $\overline{err} \approx 0$. To account for this bias, [20] suggests the .632+ bootstrap, which acts as a compromise between the .632 bootstrap and the LOO bootstrap, depending on how overfit the model is. Overall, the .632 and .632+ bootstraps are used to reduce the pessimistic bias of the simple LOO bootstrap.

As was the case for CV, there is less intuition for the variance of bootstrap error estimates. Bootstrap methods naturally estimate the variances of their error estimates using the sample variance of the bootstrap samples, as shown in Algorithm 2. Unfortunately,

in the case of the LOO bootstrap, the sample variance underestimates the variance of the error estimate, for reasons similar to why the sample variance of (repeated) K-fold CV estimates underestimates variance: the error estimates for the bootstrap samples are positively correlated with each other due to their overlapping training sets [20]. [20] thus suggests an adjusted estimate for the standard error of the LOO bootstrap.[18]

Heuristically, we can consider the variance of bootstrap error estimates in terms of the similarities between the training datasets used. Unlike K-fold CV, where each training set is guaranteed to share $\frac{K-2}{K}\%$ of all the data with each other training set, the random sampling used in the bootstrap has no such guarantees. On average, though, each bootstrap sample will share 40%[19] of its data with each other bootstrap sample, which is similar to K-fold CV for $K = 3$, and less than the overlap for $K \geq 4$. As such, the variance of bootstrap error estimates should be lower than that of most K-fold CV estimates, on average.

2.5.3 Comparisons in the Literature

A wide range of simulation studies in the literature have compared bootstrap estimates and K-fold CV estimates of model error for different models. This section will briefly survey some of the most-cited papers on this topic.[20]

[18] runs a simulation study using Fisher's linear discriminant function on very small (N = 14-20) sample sizes. It compares LOOCV with some variations of the bootstrap and concludes that the .632 bootstrap, while slightly more biased than LOOCV, particularly in overfit situations, is the best error estimator overall due to its much smaller variance. [7] compares how well the naive bootstrap, 10-fold CV, and LOOCV estimate the expected test error of OLS regression models. It finds that 10-fold CV and the naive bootstrap both outperform LOOCV for model assessment. [65] also uses OLS regression to compare the effects of different fold values for K-fold CV, finding that K-fold CV is good at estimating error for $K > 2$. Finally, [34] provides a comprehensive comparison of K-fold CV and the bootstrap using C4.5 decision trees and naive Bayes classifiers. It finds that the bootstrap

[18]Stating the estimate requires extensive notation, so in the interest of conciseness we do not do so here.

[19]The probability that observation i will be in two bootstrap samples B_1 and B_2 is approximately $P(i \in B_1 \cap i \in B_2) = P(i \in B_1)P(i \in B_2) = .632^2 = 0.4$.

[20]The interested reader can find other surveys of these kinds of papers in [23] and [34].

has low variance, but can have high bias in some applications. It recommends 10-fold CV in general, as it provides the best balance of low bias and low variance on the datasets used in the paper. These early studies generally confirm the bias-variance tradeoff of adjusting K in K-fold CV and also agree that bootstrap error estimates tend to be less variable than K-fold CV error estimates, though they are sometimes highly biased.

Later studies include comparisons with repeated K-fold CV and implement more modern learning algorithms (e.g., random forests, neural nets). Though a number of these studies exist, we highlight [33] because it focuses on the model-assessment methods discussed in this thesis and is quite comprehensive in doing so.[21] [33] compares the .632+ bootstrap with both repeated and non-repeated 10-fold CV. It uses pruned decision trees, which are good at generalizing to out-of-sample observations, and the discrete adaboost algorithm, which is highly adaptive to training data (i.e., prone to overfitting). In simulations, repeated 10-fold CV produces estimates with similar bias but less variance than single 10-fold CV, so [33] recommends the use of repeated CV. Though the .632+ bootstrap and repeated 10-fold CV both have low bias and similar variance in most cases, the .632+ is badly optimistically biased in some cases, particularly when an adaptive rule is used with large sample sizes of data. [33] thus recommends the use of repeated 10-fold CV in general.

In summary, the literature generally concludes that 10-fold CV (or repeated 10-fold CV) strikes the best balance between bias and variance for model assessment. The sensitivity of the bias of bootstrap methods to the particular experimental setup makes them less desirable to use, since model assessment values low bias.

[21] See [6], [43], and [60] for some of the other more relevant ones.

Chapter 3

Model Assessment in a Spatial Context

The model assessment methods described in Chapter 2 work well in most situations. On spatial data, however, they break down because they depend on assumptions that are often violated in spatial contexts. In this chapter, we elaborate these issues before surveying some modified CV and bootstrap methods for spatial data.

3.1 Challenges with Spatial Data

Spatial Autocorrelation

Spatial data typically exhibit *spatial autocorrelation*, where observations that are close to each other in space have related values. Figure 3.1 shows this phenomenon in a small sample from the data that will be used in Chapter 5. Note how tree species cluster together. Because of this clustering, any attributes related to species, e.g., tree size, resource consumption, etc., will also show similar spatial clustering patterns.

Spatial autocorrelation often leads to difficulties with statistical methods (e.g., see [38], [39]). Many methods assume that observations are independent and identically distributed (i.i.d.), i.e., that drawing one observation does not affect the probability of drawing another and that each observation is drawn from the same population. While this idealistic assumption is rarely true in practice, it is often close enough to the truth that it can be assumed without significantly affecting results. In spatial data, however, drawing one observation means that nearby observations will likely be similar to it, which makes it unreasonable

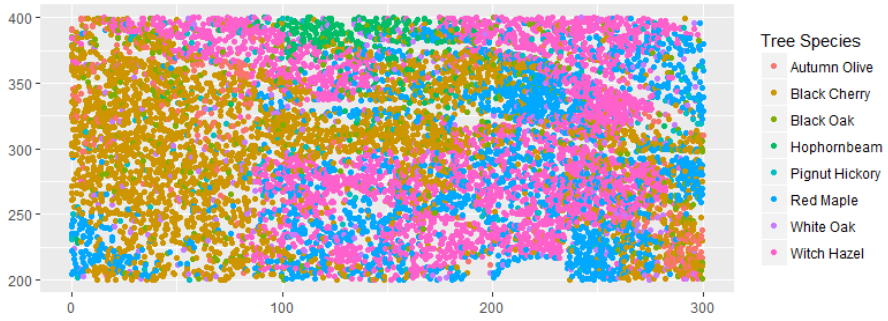


Figure 3.1: Spatial distribution of tree species

to assume that points are i.i.d. As such, many methods, including modeling and model assessment methods, perform worse when naively applied to spatial data.

Interestingly, CV and the bootstrap do not necessarily assume that observations are i.i.d. Instead, they depend on a slightly different assumption of independence based on the concepts of sampling and resampling.

Sampling and Resampling

When scientists gather data for a training set, they collect a random *sample* from the population. CV and the bootstrap aim to replicate sampling by *resampling*. Rather than sampling from the whole population, CV and the bootstrap resample from a given dataset. For example, in Figure 3.2, the blue sample in the figure in the middle figure acts as a proxy for the population from which the resamples in the rightmost figure are taken.

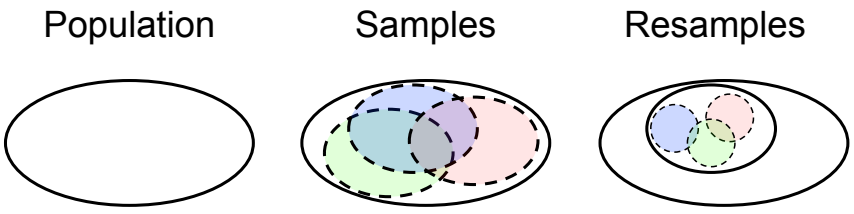


Figure 3.2: Population vs. Sampling vs. Resampling

Both CV and the bootstrap assume that the original sample is representative of the population. They also assume that their resamples "look like" samples, i.e., that the resamples

are collected in the same way that a new sample from the population would be collected. If observations are i.i.d., then this assumption is satisfied, since all points are assumed to be collected independently. Without an i.i.d. assumption, though, CV and bootstrap techniques still work for spatial data so long as the resampled training and validation sets are collected in the same manner as the original sample was: collected in spatial groups.

K-fold CV and the standard bootstrap methods described in the previous chapter, however, do not resample spatial groups of data. Instead, they use random resampling, which produces training and validation sets whose points are distinct but come from overlapping spatial regions. To illustrate this point, Figure 3.3 gives an example of how one fold of 5-fold CV and one LOO bootstrap sample might partition points into a training set (white points) and a validation set (black points).

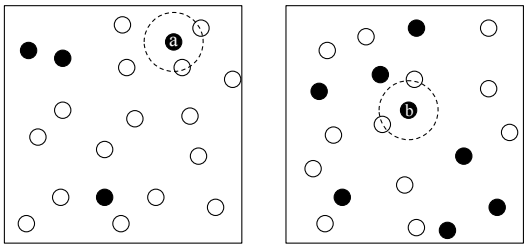


Figure 3.3: Example 5-fold CV fold and LOO bootstrap sample

Having training-set points interspersed among validation-set points leads to three main challenges for model assessment. First, the presumed independence between the training set and the validation set is broken. Consider the two validation-set points a and b in Figure 3.3 in dotted circles, which are very close to observations in their corresponding training sets. Due to spatial autocorrelation, knowing information about the nearby training-set observations gives information about a and b.[1] To clarify, suppose that a has a response y_a of the form $y_a = f(\mathbf{x_a}) + \epsilon_a$. The training-set points near a would not only tend to have $\mathbf{x_i}$ similar to $\mathbf{x_a}$, but also tend to have ϵ_i similar to ϵ_a. Thus, if a model overfit the training set, it would perform exceptionally well on a, which would undermine the rationale for using a validation set (instead of training error) in the first place.

[1] [15] notes that spatial observations can exhibit clustering behavior, which would exacerbate this problem.

The second challenge is that the spatial autocorrelation structures in the training set are not preserved. We notice that the training sets in Figure 3.3 contain observations that are more spread out in space, on average, than the original training set of all of the observations. As such, the observations do not exhibit the same degree of spatial autocorrelation as the original training set, which can affect model training.

The final challenge has to do with *interpolation* and *extrapolation*. Researchers are usually interested in assessing model extrapolation; that is, how well their models can predict observations in spatial regions other than that of the original sample. Such observations will often have different values of latent (i.e., unobserved) variables. Interspersing training-set points among validation-set points, however, leads to similarities between the latent-variable values of the training and validation sets, since they are drawn from overlapping spatial regions. Thus, K-fold CV and the bootstrap are better at assessing model interpolation: how well the models can predict observations in the same spatial region as the training set.

Challenges in the Literature

The aforementioned theoretical issues with K-fold CV and standard bootstrap methods in spatial contexts have led researchers to explore the effects of spatial autocorrelation on model assessment. The field of ecological niche modeling, the modeling of species distributions over time and space, has produced a significant amount of recent research in this area, as ecologists have discovered that failing to use proper validation techniques to evaluate their species distribution models leads to unjustified conclusions [51].

Most importantly, ignoring spatial autocorrelation falsely inflates assessments of model accuracy. Using historical data on the distributions of British breeding-bird species, [2] shows that "measures of performance on nonindependent data provided optimistic estimates of models' predictive ability on independent data." Similarly, [59] finds that the clustering of training and validation points can predict the value of the AUC[2] of popular niche models. When a spatial filter is set so that training-set observations are located closer to validation-set observations, estimates of model performance increase, all things remain-

[2]AUC is the **a**rea **u**nder the receiver operating characteristic (ROC) **c**urve. Intuitively, it provides a single numeric measure of how well a model avoids false positives and negatives. See [23] for a simple introduction, or [30] for a more comprehensive argument for using AUC instead of accuracy.

ing equal – a disturbing result, considering that the goal of model assessment is to estimate the performance of a model trained on observations drawn from the population, regardless of their particular locations. The results of a comprehensive study in [4] confirm that as the independence between training and validation sets increases, assessments of model accuracy decrease, for a variety of assessment and modeling techniques.

The studies cited above focus on validation-set approaches, but ecologists have also studied K-fold CV. [61] shows that using K-fold CV error for assessment leads to low error estimates for models that overfit the data, similar to using training error. Because CV is highly sensitive to the average distances between training and validation points [29], the proximity of training and validation data in K-fold CV leads to artificially low error estimates for models that can incorporate the information about the validation data contained in the training data. [55] supports this idea. Using data on sea temperatures in the Northern Atlantic, [55] compares K-fold CV estimates of error to extrapolation error, found by assessing model performance on sea temperatures from the Southern Atlantic. They find that K-fold CV significantly underestimates the extrapolation error for neural nets, because neural nets are flexible enough to overfit the spatial autocorrelation structures in the data.

Spatial autocorrelation also causes models to deem insignificant dependent variables as important to modeling response variables. [39] shows that even if a spatially autocorrelated variable is simulated with a spatial pattern *independent* of a spatially autocorrelated response variable, the magnitude of its correlation with the response will increase with the degree of its own autocorrelation. In context, this means that if a model assessment method ignores spatial autocorrelation, it may suggest that a model that uses non-significant yet spatially autocorrelated variables will outperform a model that uses significant but less spatially autocorrelated variables.

3.2 Methods to Account for Spatial Autocorrelation

The development of model-assessment methods that appropriately account for spatial autocorrelation is an active area of study. While these methods range from incorporating contagion terms or error-dependence structures in models (see [51], [22]) to evaluating the

effects of pairwise distances using a null model (see [29]), this thesis focuses on modifying CV and the bootstrap to account for spatial autocorrelation.

For spatial data, the resampling in CV and the bootstrap needs to mirror the original sampling, so resampled points should be spatially grouped. Figure 3.4 shows an example of this. On the left, the solid black circles represent observations that have been randomly sampled (top-left) and resampled (bottom-left). On the right, observations are sampled and resampled in spatial clusters. Note that while the sampled/resampled observations on the left are uniformly spread across the rectangle, the sampled/resampled observations on the right are closer together. This closeness preserves the spatial relationships in the data.

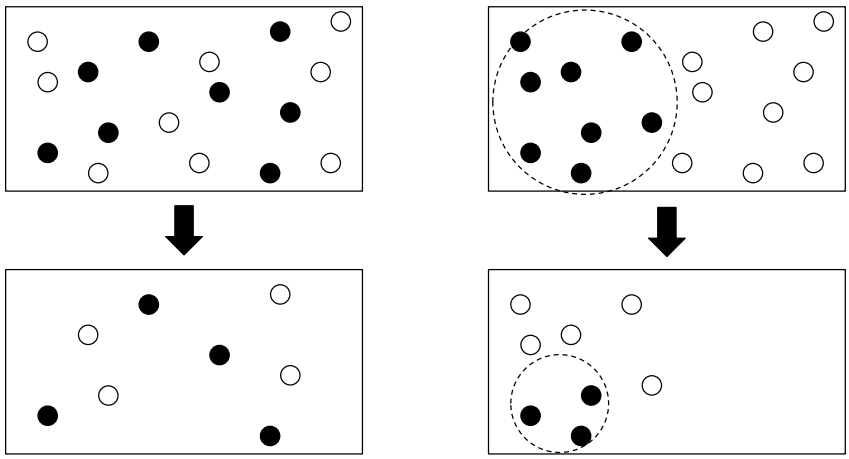


Figure 3.4: Random sampling and resampling vs. spatially grouped sampling/resampling

Grouping observations together in specific ways for CV or the bootstrap is known as *blocking* [21]. Intuitively, blocking changes the units of resampling from individual observations to user-defined blocks of observations. There are many blocking methods, depending on the needs of the given dataset. The rest of this chapter will consider spatial blocking, where blocks are groups of observations that are close to each other in space.

Blocking can help preserve the spatial relationships in the data, but it does not account for the issues of dependence and extrapolation. Observations near the boundaries of spatial blocks may still contain information about each other, due to spatial autocorrelation. To ensure that the information in the training set is independent from the information in the

validation set, we can incorporate buffer regions, which are simply points spatially located between the defined training and validation sets that are used in neither set.

Using buffer regions accomplishes two goals. First, it reduces the dependences between points near the edges of training and validation sets [37]. Second, it adds distance between training and validation sets, which increases the amount of extrapolation in a model's predictions on a validation set [21]. In summary, spatial blocking and buffering provide solutions, at least theoretically, to the issues discussed in Section 3.1. The following sections will explore how blocking and buffering can be incorporated into CV and the bootstrap.

3.2.1 Spatial Cross-Validation

Spatially blocked CV, or *spatial cross-validation*, modifies K-fold CV for spatial data. Instead of defining folds of points at random, folds are defined by spatial boundaries. The most basic case of CV spatial blocking, which we call *grid cross-validation*, defines spatial folds by dividing the data into rectangular regions. The data in each block in the resulting grid is then used as a "fold" for K-fold CV. The simplest version of grid CV is 2-fold grid CV, which simply partitions the given data into two regions for training and validation.[3]

One method for incorporating buffer regions into grid CV is described in Algorithm 3, which we call *buffered grid cross-validation* [Kim and Allen, in progress].[4] In theory, buffered grid CV should account for the issues that arise with naively applying K-fold CV to spatial data. Spatial blocking helps preserve the spatial structure in the data, and using buffers increases independence and extrapolation between training and validation sets.

Algorithm 3 Buffered Grid Cross-Validation

1: Partition data $(\mathbf{x_i}, y_i)$ into $\mathbf{r} * \mathbf{c}$ folds $F_{r,c}$ of equal spatial dimension ($\mathbf{r}$ rows, $\mathbf{c}$ columns)
2: **for** each fold $F_{r,c}$ **do**
3: `buffer` $\leftarrow \{(\mathbf{x_i}, y_i) \mid (\mathbf{x_i}, y_i) \notin F_{r',c'}, r' = \{r-1, r, r+1\}, c' = \{c-1, c, c+1\}\}$
4: `train` $\leftarrow \{(\mathbf{x_i}, y_i) \mid (\mathbf{x_i}, y_i) \notin$ `buffer`$\}$
5: `validate` $\leftarrow \{(\mathbf{x_i}, y_i) \mid (\mathbf{x_i}, y_i) \in F_{r,c}\}$
6: Train model m_k on `train`
7: Record performance p_k of m_k on `validate`
8: **return** $\frac{1}{\mathbf{r}*\mathbf{c}} \sum_{k=1}^{\mathbf{r}*\mathbf{c}} p_k$

[3] [13] and [14] provide two excellent studies of this method in the field of landslide hazard mapping.

[4] See Figure 3.5 for a visual.

Other Spatial Cross-Validation Methods

Other methods for spatial CV have been proposed in the literature. Some methods block observations in different ways. For example, more oblong rectangular blocks may be useful in many spatial applications, since climate varies more from north-to-south than from east-to-west (see [21]). The `sperrorest` package in R [8] includes spatial blocking based on K-means clustering of the data, as is used to validate landslide models in [25].

Other methods incorporate buffer regions in different ways. *Spatial leave-one-out cross-validation* (SLOO), proposed in [37],[5] is a natural spatial variation of leave-one-out CV. Rather than excluding only one observation from the training set in each fold, SLOO excludes all observations within a small buffer region of the validation observation to preserve independence. [45] proposes a variation of SLOO called *spatial K-fold cross-validation* (SKCV). SKCV is K-fold CV, except for each fold, observations within a small buffer radius of the validation set observations are excluded from the training set. To help visualize these methods, Figure 3.5 illustrates one fold each of buffered grid CV, SLOO, and SKCV (respectively). Black points are points in the validation set, gray points are points in the buffer, and white points are points in the training set for the given fold.

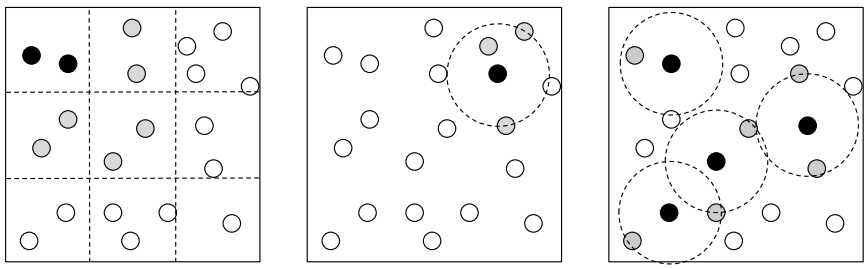


Figure 3.5: One fold of buffered grid CV, SLOO, and SKCV

The three algorithms in Figure 3.5 differ in the sizes of their training sets, validation sets, and buffer regions, which theoretically affects their bias and variance. Buffer regions are particularly interesting with respect to bias and variance. Their primary role is to increase independence between training and validation sets, which should reduce the optimistic

[5] An earlier paper, [56], proposes SLOO as well, though it does not go into the same level of detail as [37].

bias of nonspatial error estimates. Buffers also reduce overlaps between training sets by removing points from each training set, which may decrease estimate variance (see Section 2.5). At the same time, buffer regions remove data that would have otherwise been used for training, which may cause pessimistic bias. Also, the described buffering methods all remove random numbers of points, which can increase estimate variance. Since buffer regions may theoretically affect bias and variance in many ways, simulation-based results will be important to better understand their effects.

Regardless, we can make some general conclusions about the spatial CV methods we have introduced. For one, we note that SLOO is similar to LOOCV. Since it excludes the fewest buffer points and has the largest possible training set in each of its folds, it should have the lowest bias, but relatively high variance (and computational cost). The bias and variance of buffered grid CV can be tuned by $\mathbf{r}$ and $\mathbf{c}$, with high values leading to similar performance as SLOO, and low values leading to more bias but less variance. SKCV is similar to buffered grid CV. For low values of K and highly clustered points, SKCV may exclude most points from its training sets in each fold, and thus be highly biased. On the other hand, letting $K = N$ leads to SLOO.

Currently, we are unfortunately not aware of any literature that empirically compares these methods. The authors seem to propose them because they are the most natural extensions to CV for the spatial setting, but there is much room for further study.

3.2.2 Spatial Bootstrap

Like CV, the bootstrap can be modified to better deal with spatial data.[6] [26] introduces and studies *tile bootstraps*, which resample rectangular spatial tiles of points rather than individual points. The *fixed tile bootstrap* divides the original data into a grid, like grid CV, and constructs each bootstrap sample by randomly sampling B tiles, with replacement, from the resulting B tiles. The *moving tile bootstrap* constructs bootstrap samples by sampling a given number of fixed-size tiles uniformly at random over the spatial region of the original data. These tiles can either lie entirely within the boundaries of the original data or be

[6]The literature here is remarkably recent. "Bootstrap Methods and their Application," published in 1997, remarks: "Little is known about resampling spatial processes when there is no parametric model" [16].

toroidally wrapped such that, e.g., a tile that runs off the southern border of the region will continue to cover points starting from the northern border until its fixed size is reached. Toroidal wrapping helps to avoid undersampling data near the edges (see Appendix A.2). Figure 3.6 shows the differences between the different spatial bootstrap methods. Note that while the fixed tile bootstrap is likely to resample the exact same tile, that is effectively never the case in the moving tile bootstrap.

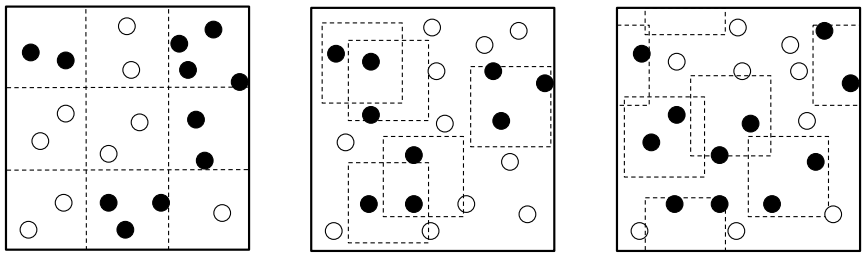


Figure 3.6: Five resampled blocks from fixed tile bootstrap, moving tile bootstrap, and moving tile bootstrap with toroidal wrapping

Bootstrap tiling has analogues in time-series bootstrap blocking, which has been the object of more study (see Appendix B). Though we do not explore time-series bootstrapping here, we note that in general, the length l of time-series blocks should be large enough that observations more than l apart are essentially independent [19]. The spatial bootstrap analogue would thus suggest that bootstrap blocks should be large enough that, e.g., observations to the east of a given block are independent of observations to its west.

Tile bootstraps help preserve the spatial autocorrelation of the original data. For model assessment, we note that the spatial bootstrap methods described in the literature do not include buffer regions, so we should have concerns about extrapolation and training/validation set independence. Regardless, incorporating spatial structure into the bootstrap makes us more confident that the error estimates produced by training models on the points within each bootstrap sample and evaluating them on the points not in the bootstrap sample are more representative of the true out-of-sample error.

We conclude this chapter with some notes on the bias and variance of the described bootstrapping methods. Returning to the intuition described in Section 2.5, we can consider

how the original sample is reused in the fixed and moving tile bootstraps. In the fixed tile bootstrap, each tile has a .632 probability of being in a given bootstrap sample, for the same reasons as provided in Section 2.5.2. Assuming that all of the tiles have similar numbers of observations within them, then, the bias of the fixed tile bootstrap should approximately match the bias and variance of grid CV with three tiles.[7] The moving tile bootstrap is more difficult to characterize in this manner. Notably, the number of blocks to resample in each bootstrap iteration affects the bias and variance; as more blocks are resampled, the probability that a given observation will be in the bootstrap sample approaches one.

We run a simulation to explore the proportion of points sampled by a moving tile bootstrap as the block size and the number of bootstrap samples changes. Details of the simulation can be found in Appendix A.2, but Figure 3.7 shows the main result.

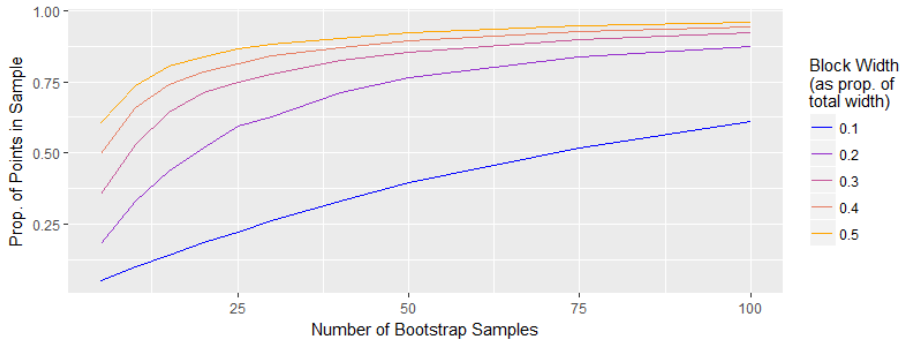


Figure 3.7: Proportion of points sampled by moving tile bootstrap, by block size and number of bootstrap iterations

As the number of bootstrap samples increases, the proportion of points sampled approaches one at different rates depending on tile size. For small tiles, the approach is nearly linear, due to the low chance of tiles overlapping. For larger tiles, the approach looks more logarithmic, with steep increases in the first few bootstrap iterations but diminishing returns as most of the points enter the bootstrap sample. The results are similar for the moving tile bootstrap with toroidal wrapping (see Appendix A.2). In context, using larger block sizes or more bootstrap iterations results in more points in each bootstrap sample on average, which in theory should lead to less bias but more variance.

[7] It seems reasonable to consider something like the .632 or .632+ bootstrap to account for this bias, but the literature on this idea is sparse and will not be explored in further detail in this thesis.

Chapter 4

Model Selection

Thus far, we have studied nonparametric resampling techniques for model assessment in both nonspatial and spatial contexts. While model assessment is useful on its own, researchers are often more interested in using it to compare models. In model selection, the goal is to choose the "best" model from a family of potential models. Ideally, the family would contain the true model that describes the real-world data-generating process. This is rarely the case in practice, though, since such processes are usually highly complex. Instead, the "best" model for a given problem is defined in one of two ways (see [3]):

1. The model expected to have the least error on new data (selection for *estimation*)
2. The simplest model that can effectively describe the data[1] (selection for *identification*)

Researchers have developed a huge variety of model selection methods for different modeling frameworks and selection goals. For example, Mallow's C_p (see [41]), Aikaike's Information Criterion (AIC, see [1]), and the Bayesian Information Criterion (BIC, see [50]) are all analytic model selection criteria that see heavy use in a wide range of applications.

This thesis, however, has focused on nonparametric model assessment methods, i.e., CV and the bootstrap, for model selection, for three primary reasons. First, it is not obvious how the theoretical justifications for the methods described above extend to spatial contexts where independence assumptions break down. For example, [3] notes that Mallow's C_p assumes that the variance of the response does not depend on predictor values, which seems

[1]The motivation for this definition comes from an idea commonly known as the parsimony principle (or Occam's razor), which suggests that the simplest explanation for some phenomenon is more likely to be the best explanation, all things remaining equal.

particularly untenable in spatial data where both predictor and response values tend to cluster. Second, extending those methods to different models is more difficult than similar extensions of CV and the bootstrap, which do not need any modification in most cases. Third, the derivations of C_p, AIC, and BIC are based on asymptotic behavior, as sample sizes $N \to \infty$. Practically speaking, the mathematical ideas behind asymptotic statistics lie beyond the level of this thesis. More theoretically, many applications do not involve large sample sizes that "approach ∞," so another layer of complexity is added in determining how quickly results converge. In general, CV and the bootstrap are much easier to understand and apply than C_p, AIC, or BIC, which makes them more appealing to use, particularly for non-statisticians. Though nonparametric methods usually involve much more computation, modern computers have made them relatively easy to run on most datasets.

Model assessment methods such as CV and the bootstrap can be used for selection for both estimation and identification, though we will see that they are perhaps better suited for the latter goal. In the literature, the first appearance of using resampling methods for model selection can be found in [24] (see [3]). Since then, CV and the bootstrap have grown to be popular methods for model selection, particularly when the models being examined are highly complex and difficult to work with on a theoretical level.

4.1 Background

Before studying resampling-based model-selection methods, we should first develop some of the concepts behind model selection for estimation and identification.

Model Selection for Estimation

The goal of model selection for estimation is to choose the model that will perform the best on new data, given a training set $\mathcal{T}$. To do so, we want to estimate the conditional test error $Err_{\mathcal{T}}$ of the candidate models and select the one with the least $Err_{\mathcal{T}}$.[2]

The theoretical criterion for selection-for-estimation methods is *efficiency* [3].[3] Using

[2]Unfortunately, as we saw in Section 2.4, CV and the bootstrap are not particularly good at estimating $Err_{\mathcal{T}}$. Regardless, they are still commonly used in selection for estimation.

[3]Not to be confused with the standard definition of statistical efficiency, which relates to the amount of data needed by an estimator to achieve some level of performance.

terminology from [3], let $\{S_m\}$ be a family of candidate models to select between, e.g., linear models with different sets of predictors. Given a training set $\mathcal{T}$, let $\hat{s}_m(\mathcal{T})$ be the family of models fit to the training set, with an optimal (i.e., least-error) model s. The goal of a model selection procedure in this context is make an estimate $\hat{m}(\mathcal{T})$ of the parameters of the optimal model, e.g., the variables used in the linear model with the least error on new data. Given a loss function L, a model selection method is asymptotically efficient if:

$$\frac{L(\hat{s}_{\hat{m}(\mathcal{T})}(\mathcal{T})) - L(s)}{min_m(L(\hat{s}_m) - L(s))} \xrightarrow[a.s.]{N\to\infty} 1.$$

In simple terms, this states that as the training set grows arbitrarily large, the loss of the model chosen by the selection procedure approaches the optimal loss. Clearly, it is best if a selection procedure can choose the least-error model as the amount of information it obtains in the training set approaches infinity.

Model Selection for Identification

The goal of model selection for identification is to choose the model that best describes underlying patterns in the population (see [33]). Such a model would produce low-error estimates not only if trained on the given $\mathcal{T}$, but also if trained on a new $\mathcal{T}'$ drawn from the same population. As such, these methods aim to minimize the expected test error Err.

The focus on expected error rather than conditional error is important here. In selection for identification, our goal is not to find the best model trained on $\mathcal{T}$. Instead, we want a parsimonious model that will perform well regardless of its specific training sample. Hopefully, such a model would provide a more accurate description of the relationships in the population, rather than an account of the relationships specific to a training set.

The theoretical criterion for selection-for-identification methods is *consistency* [3]. If s_{m_0} is the "true" model, then a model selection method is consistent if:

$$\mathbb{P}(\hat{m}(\mathcal{T}) = m_0) \xrightarrow{N\to\infty} 1,$$

i.e., if the probability that the selection method chooses the "true" model approaches one as the training set gets arbitrarily large. In general, consistency is a stronger property than efficiency, though it cannot technically be defined in frameworks without "true" models [3].

Given that CV and the bootstrap are generally better at estimating Err than $Err_{\mathcal{T}}$ (see Section 2.4), this chapter will focus on selection for identification rather than for estimation.[4]

Bias and Variance in Model Selection

Before examining the use of CV and the bootstrap for model selection, it is important to note the roles played by the bias and variance of error estimates. Model assessment prioritized low bias, since the actual values of the error estimates were important. In model selection, though, the particular error estimates are less important; instead, the ability to confidently compare error estimates becomes vital. Low variance is therefore more important than low bias to some extent. Even if the error estimates for all of the models in consideration are highly biased, CV and the bootstrap can still select the best model so long as they produce low-variance estimates and their outputs are all biased in the same way. Of course, the assumption that the estimates are "biased in the same way" for all of the models in consideration is quite strong and difficult to check. It thus may be preferable to only use CV to select between similar kinds of models (e.g., the same supervised learner with different variable inputs), since the overall behaviors of similar, non-overfit models should not change too drastically from training set to training set (assuming that the training sets are large representative samples from the population). In any case, while low-bias estimates of model error are nice for model selection, it is more important to have low-variance estimates so that the error estimates for different models can be compared to each other.

4.2 Resampling Methods for Model Selection

The CV and bootstrap methods in Chapter 2 can all be applied to model selection. Theoretically, methods with low variance should perform the best. For CV, this means that repeated K-fold CV, for $K \ll N$, should produce better results than LOOCV. In the bootstrap, the differences in the variances of the LOO bootstrap, .632 bootstrap, and .632+ bootstrap are less clear, so simulation-based results will be important to compare them.

[4]For more on selection for estimation, see [3] for an excellent (if somewhat technical) survey.

Theoretical Results

A small body of research has studied the consistency properties of K-fold CV and the bootstrap under different conditions. The most famous result is Shao's finding that for linear models, LOOCV is asymptotically inconsistent, while leave-d-out CV[5] is consistent so long as $\frac{d}{N} \to 1$ as $N \to \infty$ [52]. Shao shows a similar result for bootstrap model selection, finding that the (optimism-based)[6] bootstrap estimate of error is consistent only if the number of observations randomly drawn for each bootstrap sample is reduced from N to $m < N$, where $\{m\}$ is a sequence of integers such that $\frac{m}{N} \to 0$ and $m \to \infty$ as $N \to \infty$. In both cases, Shao shows that for asymptotic consistency, the number of observations in the validation sets must be much larger than what is typically used for model assessment. Some bias in terms of smaller training sets is traded off for larger validation sets to better account for the variation in the model assessments [66]. [65] produces similar findings for leave-d-out CV in linear regression, finding that the asymptotic probability of selecting the correct model increases as a function of $\frac{d}{N}$. [63] studies consistency in the classification setting, and finds that K-fold CV is consistent so long as $\frac{N_v}{{N_t}^2} \to \infty$ as $N \to \infty$, where N_v is the number of validation-set points, and N_t is the number of training-set points. Again, significantly more points are needed for validation to ensure consistency. In general, leaving more observations out for validation improves model selection consistency.[7]

Simulation-Based Results

The theoretical results described above generally hold true in simulation studies as well. [7] compares the model-selection performances of LOOCV, 10-fold CV, and the naive bootstrap, finding that 10-fold CV outperforms LOOCV in terms of selecting models of the right dimension. It also finds that 10-fold CV and the naive bootstrap perform similarly well on model selection in their large-sample ($N = 160$) case, though all models significantly

[5] CV with $\binom{d}{N}$ folds, where each of the $\binom{d}{N}$ possible sets of d observations is used as a validation set (i.e., excluded from a training set) in one fold.

[6] This version of the bootstrap estimates error by directly estimating the optimism (see [23], Section 7.4) of the bootstrap assessment procedure. A clear description of the method can be found in [18].

[7] Though it is not always the case that the size of the validation set needs to be much larger than the size of the training set - see [64] for an example where for comparing nonparametric regression methods, consistency can be obtained even when the training and validation sets are of the same magnitude.

underestimate the true errors of the final model they produce. [65] compares 2-fold, 5-fold, 10-fold, and LOO CV on linear models on small samples ($N = 20$). Interestingly, it finds that 2-fold CV performs worse than the other methods; asymptotic results do not entirely hold in the small-sample case.[8] [52] compares LOOCV with other CV methods (not covered in this thesis), and finds that all methods have negligible probabilities of selecting under-parameterized models. LOOCV also performs more poorly than the other methods that leave out more observations. [53] conducts a similar study comparing different numbers of observations to resample for the naive bootstrap. Like the CV procedures, the bootstrap is unlikely to pick models with fewer parameters than the true model. In a simulation using $N = 40$ observations, [53] finds that resampling only 15 observations per bootstrap sample outperforms resampling 40, which supports asymptotic results.

Notably, the literature for both the theoretical and simulation-based results discussed here is somewhat dated, and the simulation-based results are run with sample sizes far smaller than would be expected in modern applications. Also, none of the studies consider the .632 or .632+ bootstraps. Unfortunately, more recent literature on the theoretical and empirical performances of the model-selection methods of Chapter 2 is sparse.

Before concluding this section, one interesting recent result on the use of CV for model selection should be highlighted. In [32], Jung draws on the ideas from Shao's papers ([52], [53]) to propose *multiple predicting cross-validation* (MPCV) as alternative to K-fold CV for model selection. Like K-fold CV, MPCV first randomly partitions the given data into K folds. For each iteration, though, rather than use $K - 1$ folds for training and one fold for validation, MPCV uses one fold for training and the other $K - 1$ for validation. The $K - 1$ error estimates for each observation are averaged in the final error estimate. Jung proves the consistency of MPCV (for linear models), though he notes that it should only be used when one fold contains enough data to provide a "reasonable fit" for the given model. He also shows how MPCV outperforms K-fold CV at model selection for a variety of both simulated and real datasets, since it has much less of a tendency to select overfit models. Jung's idea of "flipping" training and validation sets to achieve better model selection properties seems

[8] The relatively large variance of 2-fold CV (see Footnote 15 in Chapter 2) may be a factor here as well.

like an interesting idea to explore with a variety of different CV and bootstrap methods.[9] In any case, Jung's result shows that there is room in the literature for further theoretical and empirical studies of the model-selection properties of resampling-based assessment methods.

4.3 Model Selection in a Spatial Context

The spatial CV and bootstrap methods discussed in Chapter 3 can be applied to model selection. In theory, the same bias-variance considerations as at the end of Section 4.1 apply: while low bias is desirable, the best methods should be those with the lowest variance. The bias-variance discussions in Section 3.2 therefore help us hypothesize which spatial methods should perform the best at model selection. For the spatial CV methods, buffered grid CV with a coarse grid or SKCV with low K should outperform SLOO at model selection, due to their lower variance. Similarly, for the spatial bootstraps, using moving tile bootstraps with small numbers of bootstrap samples should lead to better selection properties.

As was the case for nonspatial, resampling-based methods for model selection, though, there is sparsity in the literature that specifically tackles the problem of model selection using resampling methods on spatial data.[10] Interestingly, a highly cited 2004 review of model selection methods for ecology does not mention any nonparametric resampling methods [31]. It seems that while spatial resampling for model assessment is growing in popularity (see e.g., the papers in Section 3.1), many researchers still prefer to use model-selection methods that are more theoretically supported (albeit under assumptions that are not necessarily satisfied by the data) and less computationally intensive. In general, there appears to be significant opportunity to study how the spatial model assessment methods described in Chapter 3 perform on the problem of model selection with spatial data.

[9] To explore this idea, we conduct a simple simulation study comparing K-fold CV and buffered grid CV to their "flipped" variants in Appendix A.3, based on the simulation framework in Chapter 5.

[10] There is, however, a body of research on CV model selection for dependent time-series data (which is similar to one-dimensional spatial data, in some sense). Some of this research is surveyed in Appendix B.

Chapter 5

Simulations

Chapters 3 and 4 gave heuristic justifications for why spatial blocking and buffering should improve resampling-based assessment and selection on spatial data. As noted then, however, few studies have empirically compared such techniques to other methods.[1] This chapter takes some steps toward addressing this gap in the literature, though there is much more work to be done. Using a simple simulation framework, we run some basic comparisons of spatial versus nonspatial resampling techniques on both simulated and real-world spatial data, and conclude with some preliminary results and guidelines for future extensions.

5.1 Simulation Studies

Our simulations aim to show that for spatially structured data, spatial resampling methods indeed outperform nonspatial methods at model assessment and selection. To do so, we adopt simplified versions of the simulation setup used in Box 1 of [21]. Though we relegate details to Appendix C, Sections 5.1.1 and 5.1.2 outline our methods and results.

5.1.1 Model Assessment Simulation

The model assessment simulation compares the true errors of linear models to the error estimates produced by six[2] methods: training error, K-fold CV error, LOO bootstrap error,

[1] There is some literature on these kinds of comparisons, e.g., the papers in Section 3.1 on the overoptimism of nonspatial methods on spatial data. Even then, [21] notes that while spatial CV methods are indeed more pessimistic than nonspatial methods, "few studies have explicitly demonstrated that the estimates resulting from blocked cross-validations are indeed closer to the 'true' error."

[2] See Appendix C for why we chose to use these six estimates.

buffered grid CV error, SLOO CV error, and moving circle[3] bootstrap error. The process is as follows. First, we generate 100 datasets, each a 100x100 grid of 500 points in space. For each point, we randomly generate spatially autocorrelated variables `X1`, `X2`, and `X3` along with spatial noise `e_spatial` using multivariate normals with covariance matrices based on the pairwise distances between points (details in Appendix C). We then generate the response `growth` as $f(\texttt{X1}, \texttt{X2}, \texttt{X3}) + \texttt{e\_spatial} + \texttt{e}$, where `e` is nonspatial random noise.

For each simulated dataset $\mathcal{T}_i$, $i = 1, 2, \ldots, 100$, we compute the aforementioned six estimates of the error E_k, $k = 1, \ldots, 6$ of the linear model m: $\texttt{growth} \sim \texttt{X1} + \texttt{X2} + \texttt{X3}$. We also compute the true (conditional[4]) error for each $\mathcal{T}_i$ by fitting m to $\mathcal{T}_i$ and averaging its error over all 99 other datasets $\mathcal{T}_j$, $j \neq i$. Intuitively, we treat the other 99 datasets as a large draw from the same population as $\mathcal{T}_i$, so performance on that large sample should be a good estimate of m's performance on the population in general.

Table 5.1 shows the average values of the six error estimates over the 100 simulated datasets. Training error is the most optimistic assessment method, as expected. Also, the

Assessment Method	Average Error Estimate
Training Error	1.666
K-fold CV	1.683
LOO Bootstrap	1.686
Buffered Grid CV	1.766
SLOO CV	1.728
Moving Circle Bootstrap	1.741

Table 5.1: Average error estimates for six assessment methods

nonspatial resampling methods are more optimistic than the spatial resampling methods.

To compare these error estimates to true error, we plot the differences between the error estimates and true error (Figure 5.1). All six assessment methods are overoptimistic: the peaks of their distributions lie to the left of 0. The spatial error assessments in red, though, tend to be less optimistic than the nonspatial assessments in black. In other words, spatial

[3]Instead of sampling rectangular regions like a moving tile bootstrap, circular regions are sampled.

[4]We could produce an estimate of true expected error for the linear model $\texttt{growth} \sim \texttt{X1} + \texttt{X2} + \texttt{X3}$ by averaging all 100 conditional errors. We do not explore the differences between conditional and expected error, though such analyses may be interesting to look into in the future.

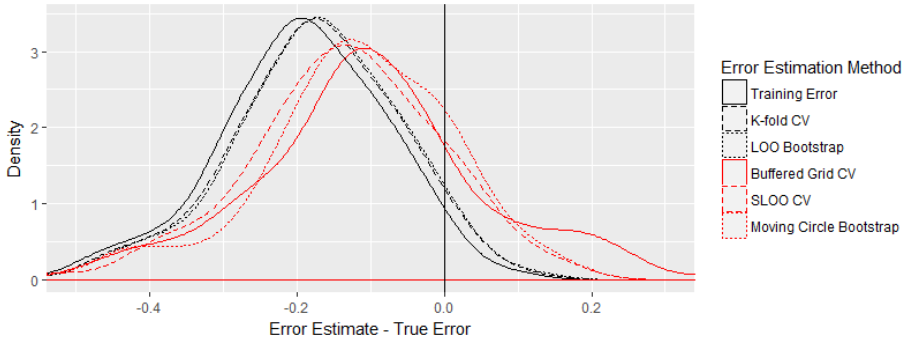


Figure 5.1: Differences between error assessments and true error

resampling methods tend to produce error estimates closer to the true error.

Table 5.1 and Figure 5.1 suggest that, at least for the simple simulation presented here, spatial resampling methods indeed outperform nonspatial ones at model assessment. We confirm not only that spatial methods are less overoptimistic, but also that their estimates actually tend to be closer to true error than the estimates made by nonspatial methods.

5.1.2 Model Selection Simulation

We now consider whether the better model assessment properties of spatial methods translate into better model selection. The simulation setup here is identical to the previous simulation, with two exceptions. First, we generate `growth` using $f(\texttt{X1}, \texttt{X2}) + \texttt{e\_spatial} + \texttt{e}$. Second, for each dataset we compute the six error estimates E_k for four models:

1. m_1: $\texttt{growth} \sim \texttt{X1}$ (underspecified model 1)
2. m_2: $\texttt{growth} \sim \texttt{X2}$ (underspecified model 2)
3. m_3: $\texttt{growth} \sim \texttt{X1} + \texttt{X2}$ (the true model)
4. m_4: $\texttt{growth} \sim \texttt{X1} + \texttt{X2} + \texttt{X3}$ (the overspecified model)

We say that each model-assessment method k "selects" the model m_j for $\underset{j}{\operatorname{argmin}}\ \mathrm{E}_k(m_j)$, i.e., the model that the model-assessment method k estimates to have the least error.

Table 5.2 shows the number of times each model was selected by each model-assessment method. Ideally, we want our selection methods to choose the true model m_3. Spatial methods select m_3 more often than do nonspatial methods. Also, none of the methods predominantly select the underspecified models, which matches our expectations based on the results of Section 4.2. Interestingly, buffered grid CV and SLOO are less likely to select

Selection Method	Selected m_1	Selected m_2	**Selected $\mathbf{m_3}$**	Selected m_4
Training Error	0	0	**0**	100
K-fold CV	0	2	**47**	51
LOO Bootstrap	0	1	**37**	62
Buffered Grid CV	0	13	**62**	25
SLOO CV	0	6	**69**	25
Moving Circle Bootstrap	0	8	**51**	41

Table 5.2: Models selected by six selection methods

the overspecified model m_4 than are the other methods. We suspect that this may be due to buffer regions. m_4 contains `X3`, a spatially autocorrelated variable with no defined relation to the spatially autocorrelated `growth`. According to [39], though, even if `X3` has a spatial pattern independent of `growth`, it will be more correlated with `growth` than a nonspatial variable `X4` would be. Because training error, K-fold CV, etc. do not use buffer regions, their training sets contain points near their validation sets. Thus, they are more likely to capture the spurious relationships between `X3` and `growth`, which would make them more likely to assess m_4 as being better than m_3.

In conclusion, our simulations show that spatial resampling techniques outperform nonspatial ones at both model assessment and selection. Though the improvements are not dramatic, they are a promising foundation for future work in this area (see Section 5.3).

5.2 Study on Real-World Data

Finally, we apply the model-assessment methods to a real-world dataset collected by Albert Y. Kim and David Allen. Details of the dataset and models are relegated to Appendix C. We divide the spatial region of the data into six "strips" for the study, as illustrated in Figure 5.2.[5] For each strip, we estimate model error using five[6] model-assessment methods. Table 5.3 shows the average error estimates from the five assessment methods. Again, we see that spatial methods are less optimistic than nonspatial methods.

A more in-depth exploration complicates this picture. Figure 5.3 shows how the models

[5]We choose these particular strips only because they intuitively divide the data into a few large regions.

[6]We do not use SLOO because it is too computationally expensive (see Appendix C).

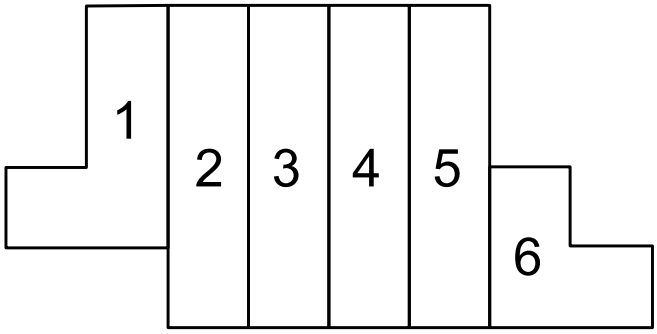


Figure 5.2: Division of study region into six strips

Assessment Method	Average Error Estimate
Training Error	0.141
K-fold CV	0.142
LOO Bootstrap	0.142
Buffered Grid CV	0.146
Moving Circle Bootstrap	0.148

Table 5.3: Average error estimates for five assessment methods

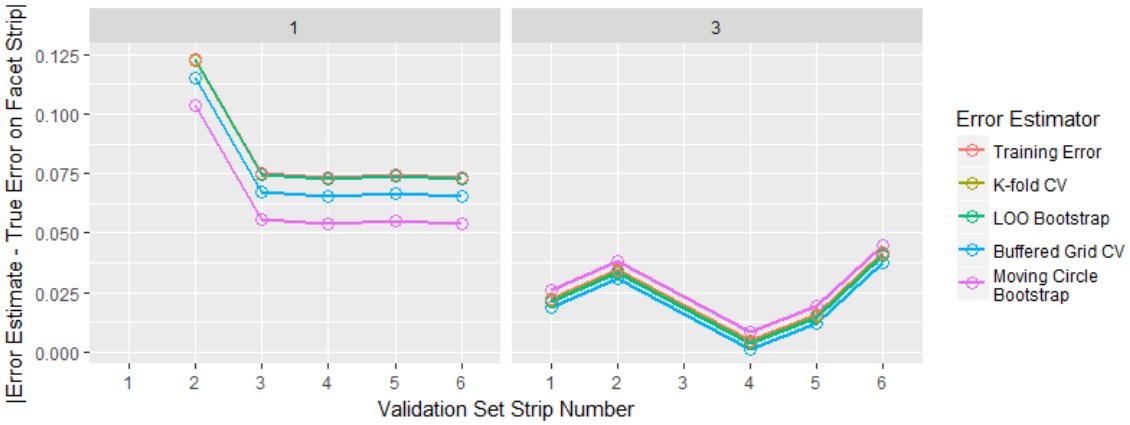


Figure 5.3: Performances of assessment methods for strips 1 and 3

trained on strips 1 and 3[7] perform on the other five strips, where we compute "true" error by training the model on the given strip and taking its average error on the other five strips. For strip 1, we see that the moving circle bootstrap and buffered grid CV outperform the nonspatial methods, which produce essentially the same estimates as each other. For strip

[7]Appendix C contains the results for strips 2, 4, 5, and 6.

3, buffered grid CV still performs well, but the moving circle bootstrap performs worse than the nonspatial methods. We highlight strips 1 and 3 because they represent the mixed results of the study. Table 5.3 shows how spatial methods are more pessimistic than nonspatial methods, which generally leads to estimates closer to true error. In cases such as strip 3, however, the spatial methods can be too pessimistic, which leads the nonspatial methods to be closer to true error. Of course, real-world data are messy, and we take a very basic approach to modeling the given dataset. There are significant opportunities for more detailed study here.

We run the same model selection study as in Section 5.1.2, using the six models shown below (see Appendix C for explanations of the variables):

1. m_1: `growth` $\sim$ `dbh` + `ncomp`
2. m_2: `growth` $\sim$ `dbh` + `biomass_comp`
3. m_3: `growth` $\sim$ `dbh` + `biomass_comp` + `ncomp`
4. m_4: `growth` $\sim$ `species` + `dbh` + `ncomp`
5. m_5: `growth` $\sim$ `species` + `dbh` + `biomass_comp`
6. m_6: `growth` $\sim$ `species` + `dbh` + `biomass_comp` + `ncomp`

Table 5.4 shows the results from the model selection simulation on the real data.

Selection Method	m_1	m_2	m_3	m_4	m_5	m_6
Training Error	0	0	0	0	0	6
K-fold CV	0	0	0	1	2	3
LOO Bootstrap	0	0	0	1	2	3
Buffered Grid CV	0	0	0	1	3	2
Moving Circle Bootstrap	0	0	0	1	3	2

Table 5.4: Models selected by six selection methods

For the real data, we do not know the "true" model. Our results, though, indicate that it may be m_5, since it is selected the most often by the spatial resampling methods.[8]

[8]We do not explore how one could determine the "significance" of the differences between error estimates to more confidently use resampling techniques for model selection, though it is an interesting area of study.

5.3 Recommendations for Next Steps

The results of this chapter can be extended in many ways. This section provides some guidelines and recommendations for future study.

Keep it Simple

There are a huge number of parameters that one can control in a factorial simulation setting. A (non-exhaustive) list of our parameter choices includes:

- The particular resampling methods being tested: we chose training error, K-fold CV, the LOO bootstrap, buffered grid CV, SLOO CV, and the moving circle bootstrap.[9]
- The level of noise and spatial autocorrelation: see Appendix C for the values we used.
- The density of points in the sample: we used 500 points in a 100×100 grid.
- The spatial distribution of points: we simulated points uniformly at random across the sample regions, without clustering.
- The shapes of the regions: we only considered square regions.
- Different kinds of signals: see Appendix C for the signal we used.
- Different kinds of spatial noise: we use a random spatially autocorrelated variable.[10]
- Different models: we used multiple linear regression models.

Studying the effects of systematically tuning any (or all) of these parameters would produce interesting results; however, we suggest that such studies examine the effects of tuning only one or two parameters at a time, since tuning many parameters at once leads to factorial growth in the number of parameter combinations to try.

Don't Keep it too Simple

An important fact to keep in mind is that the ideas in this thesis only hold up if the signal being modeled is complex enough to require a significant amount of data to learn. If the signal is too simple, results will be inconsistent. To account for this fact, we suggest

[9] Appendix A.3 also tests "flipped" CV methods, where models are trained on the smaller validation sets and validated on the larger training sets.

[10] One example of another possibility here would be to use a simple linear "slope" variable that decreases in a given direction. This suggestion is motivated by the results of [4], which suggest that interpolation effects (e.g., predicting on a region between points in the training set) may play significant roles in assessment and selection.

consulting learning curves as one changes different simulation parameters. The models used in the simulations should always require significant amounts of training data before minimizing their validation errors.

Hypothesize before Simulating

Simulation settings make it easy to explore the effects of changing different parameters. Still, results can often be difficult to interpret, and may take a significant amount of time to compute. Before simulating, we suggest that there should be a clear idea of how the values of the parameter should affect results. This acts not only as a sanity check, to ensure that there are no mistakes in the simulation, but also as a way to produce more interesting insights when simulation results do not match expectations.

Chapter 6

Conclusion

In this thesis, we explored the properties of CV and the bootstrap for model assessment and selection. We began by introducing CV and bootstrap methods as statistical estimators of model error, noting the statistic that they estimate (Err vs. $Err_{\mathcal{T}}$) as well as their bias and variance. We then extended CV and the bootstrap to spatial settings. First, we showed how nonspatial CV and bootstrap methods fail in spatial settings because they fail to produce independent training and validation sets, preserve spatial structures within training sets, and induce appropriate levels of extrapolation. To account for these issues, we incorporated blocking and buffering into CV and the bootstrap.

We then moved on to the problem of model selection. After covering some theoretical background, we showed how assessment techniques could be used for model selection, paying particular note to the roles of bias and variance in assessment vs. selection. We proceeded to survey theoretical and simulation-based studies on selection.

Finally, we performed some simple simulation studies. Using simulated spatial data, we showed that spatial resampling techniques outperform nonspatial techniques at both assessment and selection. Our results using real-world data were less clear-cut, but in general our simulations indicated that it does make sense to use spatial methods for spatial data. We concluded our simulations with some suggestions for future extensions of our project.

As statistical modeling grows in popularity, it becomes increasingly important to better understand model assessment and selection. In many situations, such as the spatial contexts explored in this thesis, data do not follow naive assumptions of independence. Failing to

account for dependencies can lead to a variety of consequences: practitioners may act on false beliefs that their models are better than they actually are, and researchers may incorrectly conclude that overspecified models best describe the underlying processes that they study.

This thesis lays a foundation for understanding and exploring assessment and selection for spatially dependent data, though there is clearly much more work to be done. Dependent data are ubiquitous, and it is often difficult to come up with methods to deal with them in model assessment and selection. A better understanding of how to deal with spatial dependencies in these situations may be a key step in the right direction.

Appendix A

Miscellaneous Simulations

A.1 *Elements of Statistical Learning* Replication

The end of Chapter 7 in [23] details a simple simulation to "examine the question of whether cross-validation does a good job in estimating $Err_{\mathcal{T}}$... as opposed to the expected test error." The simulation setup is as follows. 100 datasets are simulated, where each dataset is 80×21: eighty observations, each with twenty variables X_i, $i = 1, 2, \ldots 20$ uniformly distributed on $[0, 1]$ and one response variable that is 1 if $\sum_{j=1}^{10} X_j > 5$ and 0 otherwise. The simulation considers a best-subset linear regression model with various values of p, the number of predictors to choose for the model.

In the first simulation, [23] plots the mean absolute deviation between $Err_{\mathcal{T}}$ and the error estimates produced by LOOCV and 10-fold CV, for each subset size p. The "true" $Err_{\mathcal{T}}$ is computed by training the model on the given sample and evaluating it on a "large test sample." Since the size of this sample is not specified, we use a sample of size 1,000 as our "large test sample." In the second simulation, [23] plots the correlation between $Err_{\mathcal{T}}$ and the error estimates produced by LOOCV and 10-fold CV, for each subset size p. We replicate both simulations, but include results from the naive bootstrap, the LOO bootstrap, and the .632 bootstrap in addition to the results from 10-fold CV and LOOCV.

In the first simulation (Figure A.1), we exclude the results for LOOCV for readability, since they are essentially identical to the shown results for 10-fold CV. Other than that, the simulation reveals a few interesting details. First, we notice that the simple bootstrap performs the worst at estimating error and the LOO bootstrap performs the best, with the

performances of the .632 bootstrap and 10-fold CV somewhere between those two extremes. At high values of p, though, this behavior changes. Recall that the true model has $p = 10$ predictors; in other words, most of the predictors chosen after the first 10 predictors should be random noise. While the inclusion of noisy predictors does not affect the predictions made by CV or the .632 bootstrap, it hurts the predictions made by the simple and LOO bootstraps. This result indicates that the .632 bootstrap may be more useful than the other bootstrap methods for model assessment in applied settings, where the complexity of the true model is rarely known.

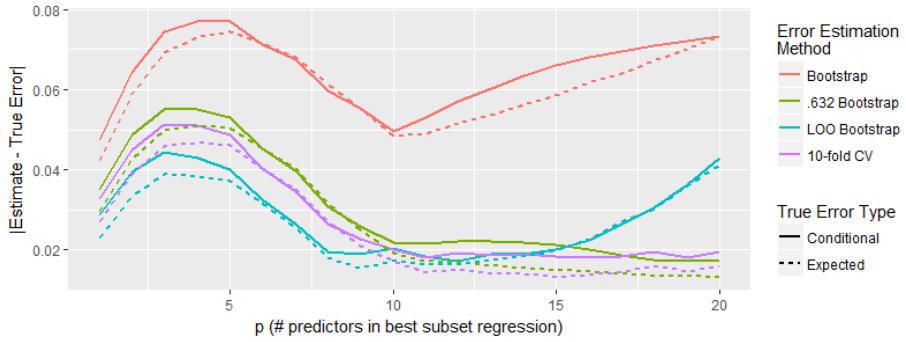


Figure A.1: How well error-estimation methods estimate conditional and expected error

The second simulation (Figure A.2) simply plots correlation versus p. We notice that average correlation is consistently around -0.2, except for $p = 4$ through $p = 9$, where it spikes up to 0. This result differs slightly from the result in [23], where the correlation is around -0.2 except for $p = 6$ through $p = 14$, where it spikes up to 0.2. No explanation for this phenomenon is provided in [23], and we do not attempt to produce one here. The overall conclusion is that the exact error estimates produced by CV and the bootstrap are very weakly correlated with conditional error.

A.2 Spatial Bootstrap Simulation

We run 100 iterations of the bootstrap simulation. In each iteration, we first generate 1000 points uniformly at random over a 100×100 grid. Then, we record the proportion of these points that are contained in moving-tile bootstraps, gridsearching over square tile widths of 10, 20, 30, 40, and 50 units, and numbers of bootstrap iterations of 5, 10, 15, 20, 25, 30,

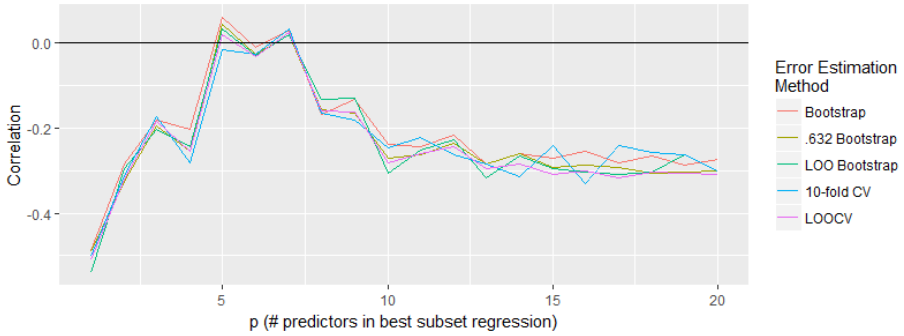


Figure A.2: Correlation between error estimates and $Err_{\mathcal{T}}$

40, 50, 75, and 100. Finally, we average the results over all 100 iterations.

Besides the main results as shown in Figure 3.7, there are two other results to note. Figure A.3 shows the standard deviation (over the 100 iterations) of the number of points in the bootstrap simulations. The results are as expected. Clearly, when nearly the entire sample is contained in the bootstrap samples, the standard deviation approaches zero. We notice in the smallest block widths, though, a sort of "tipping-point" behavior. In blocks of width 10 and 20, we see that the standard deviation increases until over 50% of the points are contained in each bootstrap sample (seen by reference to Figure 3.7), where it tapers off and begins to decrease again.

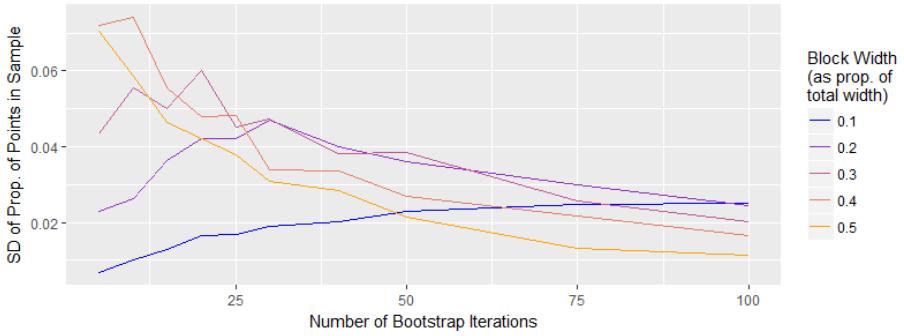
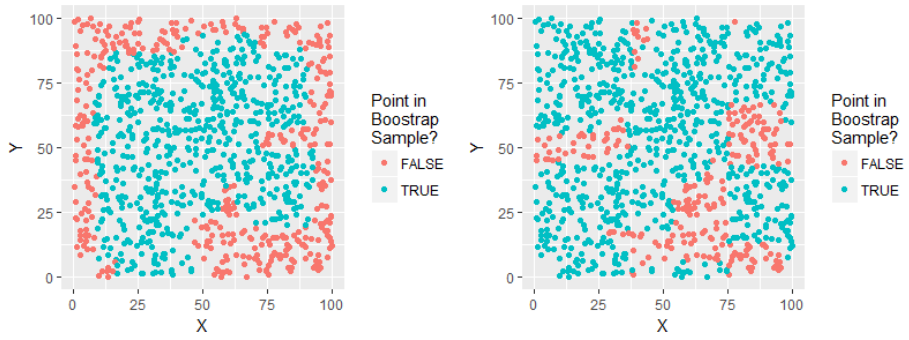


Figure A.3: Standard deviation of number of points in moving tile bootstrap

This phenomenon relates to the second result to note, which is the systematic undersampling of certain points. The simulation runs moving tile bootstrap samples without toroidal wrapping. This means that tiles are less likely to contain points near the edges of the sample region. For example, Figure A.4(a) shows this undersampling in one example,

where 15 bootstrap tiles of width 30 are sampled from the simulation region. Clearly, the points toward the edges of the region are less likely to be sampled. Thus, any bootstrap sample that contains over 50% of the points in the region will be similar to any other such bootstrap sample, since they will both contain most of the points in the middle portions of the region.



(a) No toroidal wrapping (b) With toroidal wrapping

Figure A.4: Moving tile bootstrap without toroidal wrapping undersamples edge points

In contrast, Figure A.4(b) shows 15 bootstrap tiles of width 30 sampled with toroidal wrapping. Notably, there is no undersampling near the edges. The utility of toroidal wrapping depends on the particular application of the moving tile bootstrap. In general, though, this simulation suggests that one should exercise caution when interpreting the results of a moving tile bootstrap without toroidal wrapping, since the bootstrap samples may all be highly similar to each other.

As can be seen in Figure A.5, toroidal wrapping does not significantly impact the number of points in each moving tile bootstrap sample, though it does decrease the number of bootstrap iterations needed to sample all of the given points. With regard to the standard deviation of the number of points in the bootstrap samples, Figure A.6 shows similar "tipping-point" behavior as does Figure A.3, though the tipping point is now at about 75% of the original sample, and not 50%.

In conclusion, the spatial bootstrap simulations show the importance of selecting appropriate values for the block width and the number of bootstrap iterations to use for each bootstrap sample. Choosing values that are too high can easily lead to bootstrap samples

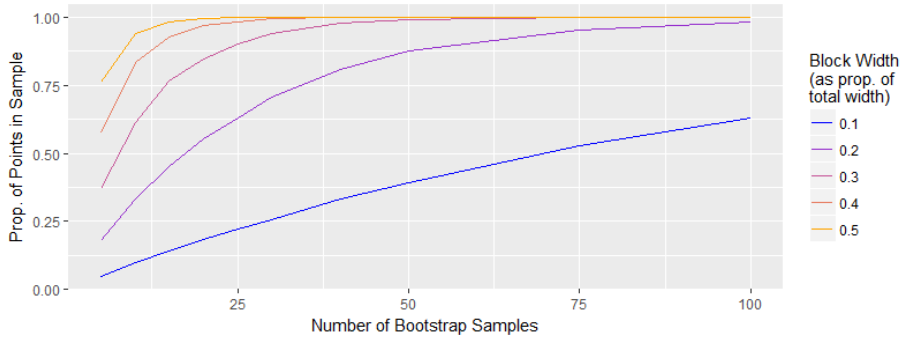


Figure A.5: Proportion of points sampled by moving tile bootstrap with toroidal wrapping

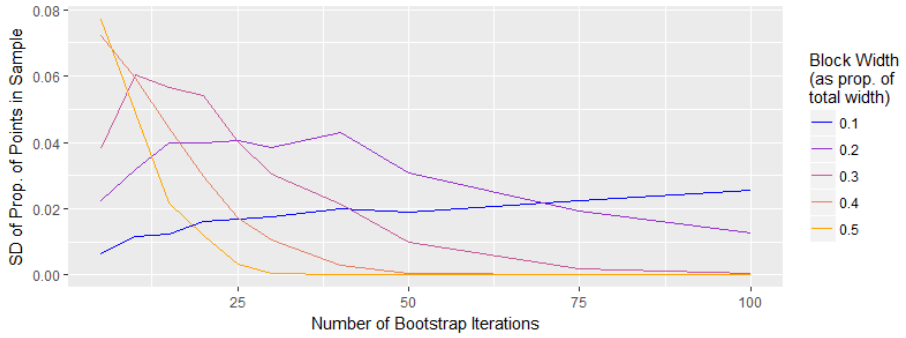


Figure A.6: Standard deviation of number of points in moving tile bootstrap with toroidal wrapping

that are highly similar to each other (particularly without toroidal wrapping), which can affect the interpretation of the results from the bootstrap.

A.3 "Flipping" CV methods

We apply the simulation frameworks established in Chapter 5 and Appendix C to comparing K-fold CV and buffered grid CV to their "flipped" variants,[1] where roles of the training and validation sets in the standard CV procedures are reversed: models are trained on the smaller validation set and validated on the larger training set.

We run these simulations for two main reasons. First, [32] shows that MPCV outperforms K-fold CV at selection for independent data. While spatial data are obviously different from independent data, the main idea behind MPCV - that using a large validation set (instead of a large training set) improves model selection performance - seems

[1]We will refer to these methods as MPCV and flipped buffered grid CV (FBGCV), respectively.

like it could apply to any kind of data, spatial or nonspatial. The second reason is an intuitive expectation for smaller training samples to improve assessment performance on spatial data. Models fit to small training sets are more likely to be overfit to those specific training data. If validation sets are independent from training sets (as is hopefully the case in FBGCV), however, this overfitting should lead to worse performance on the validation sets, i.e. less optimistic assessments of model error. Since the results of Chapter 5 show that the standard assessment methods tend to be too optimistic, the "flipped" methods may prove to be better at model assessment, which would hopefully translate into better model selection as well.

A.3.1 Model Assessment Simulation

Using a simulation setup identical to the one used in Section 5.1.1, we compare K-fold CV, MPCV, buffered grid CV, and flipped buffered grid CV (using $K = 10$ and a 4×4 grid). Table A.1 shows the average error estimates produced by the four methods. While

Assessment Method	Average Error Estimate
K-fold CV	1.706
MPCV	1.707
Buffered Grid CV	1.789
FBGCV	1.904

Table A.1: Average error estimates for four assessment methods

K-fold CV and MPCV produce similar estimates (on average), FBGCV produces much more pessimistic error estimates (on average) than any of the other methods.

To see how these error estimates compare to true error, we examine Figure A.7. For the simulated spatial data, K-fold CV and MPCV produce essentially indistinguishable estimates. FBGCV, though, produces estimates that are markedly less optimistic than buffered grid CV. Indeed, the distribution of error estimates produced by FBGCV appears to be centered at 0, i.e. FBGCV appears to be unbiasedly estimating true error. It does, though, have significant variance, as shown by its lower distribution peak.

The variance of FBGCV is interesting, given that "flipping" training and validation sets eliminates overlap between training sets. Our hypothesis is that due to the spatial

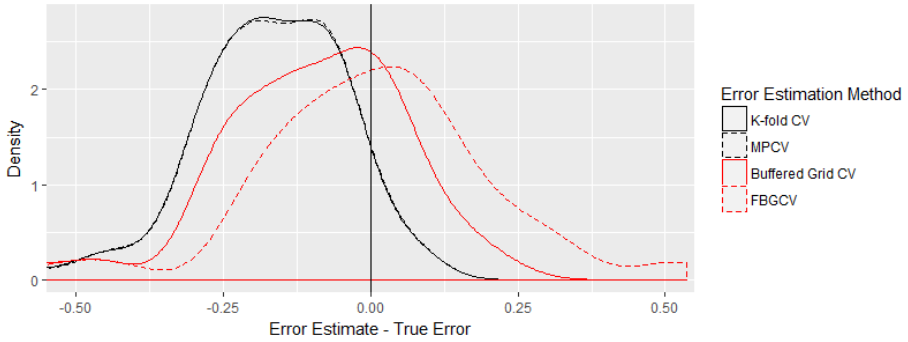


Figure A.7: Differences between "flipped" error assessments and true error

autocorrelation in the data, few of the sixteen blocks used for training in 4×4 FBGCV are highly representative of the data in general. As such, training models on single blocks leads to drastically different models in each fold, which would lead to drastically different model performances on each validation set.

We reproduce results using 5000 simulated observations per dataset (rather than 500) to ensure that the relatively small training sets used in MPCV and FBGCV are not significantly biasing results. The average error estimates for the four methods remain similar to the averages in Table A.1, so we do not reproduce that table. Figure A.8 shows the distributions of the errors. Using more observations per dataset does not significantly affect any of the

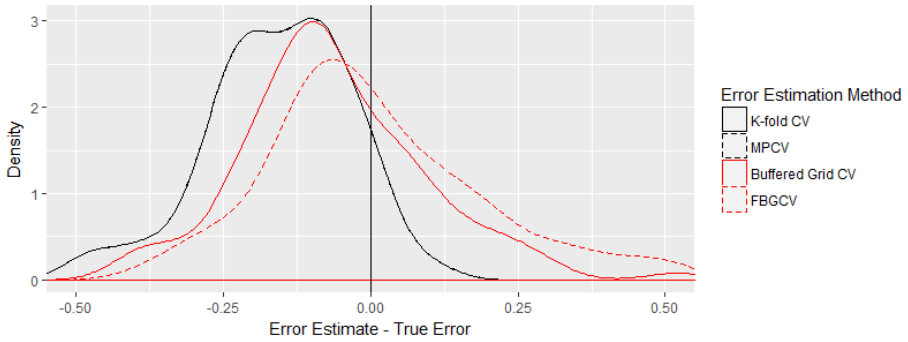


Figure A.8: Differences between "flipped" error assessments and true error, $N = 5000$

error distributions, so we conclude that 500 observations is sufficient for both MPCV and FBGCV to optimize performance.

In summary, for model assessment on spatial data, K-fold CV and MPCV both tend to be too optimistic. For the buffered grid CV methods, FBGCV appears to produce an

almost unbiased estimate of true error, though its estimates are more variable than the estimates produced by buffered grid CV. This variance makes us suspect that FBGCV may not outperform standard buffered grid CV at model selection. We test our suspicion in the next section.

A.3.2 Model Selection Simulation

Now, using a simulation setup identical to the one used in Section 5.1.2, we compare K-fold CV, MPCV, buffered grid CV, and flipped buffered grid CV (using $K = 10$ and a 4×4 grid). Table A.2 summarizes the models selected by each method for each of the 100 simulations (see Section 5.1.2 for descriptions of the four models). Model m_3 is the true model that we want our methods to select. As expected, buffered grid CV performs better than FBGCV at

Selection Method	Selected m_1	Selected m_2	**Selected $\mathbf{m_3}$**	Selected m_4
K-fold CV	0	2	**46**	54
MPCV	0	1	**47**	52
Buffered Grid CV	0	7	**67**	26
FBGCV	0	14	**58**	28

Table A.2: Models selected by six "flipped" selection methods

model selection; the variance of FBGCV estimates hurts FBGCV's performance at selection. In the spatial context of this simulation, we note that the performances of MPCV and K-fold CV are essentially identical. While MPCV outperforms K-fold CV at model selection using independent data [32], it does not seem to do so when data exhibit spatial dependencies.

In conclusion, "flipping" training and validation sets for CV does not improve model selection performance on our simulated datasets.[2] While FBGCV produces less biased estimates of true error, its higher variance causes it to perform worse at model selection than buffered grid CV. While the results here are discouraging for model selection, we note that our simulation setup is quite naive. There is still a lot of room to study and experiment with different "flipped" methods in different settings.

[2]We do not apply MPCV and FBGCV to the real-world dataset because the resulting training sets are too small.

Appendix B

Time-Series Resampling Methods

This appendix briefly discusses the literature on time-series resampling for model assessment and selection. Our goal here is simply to cover the basics of some key papers in the field for the reader interested in exploring further.

B.1 Time-Series Bootstrapping

The tile bootstraps introduced in Section 3.2.2 have analogues in time-series bootstrap methods, which have been the object of more study (at least for parameter estimation). The fixed tile bootstrap is similar to the method proposed by [10], which divides time-series data into non-overlapping, fixed-length blocks in chronological order before resampling from those blocks. The moving tile bootstrap without toroidal wrapping is more related to the *moving blocks bootstrap* (see [35] and [40]). The moving blocks bootstrap samples fixed-length chronological blocks uniformly at random from the given data. The idea of toroidal wrapping can be found in the circular moving blocks bootstrap (see [46]), which reduces undersampling from the beginning and end of the data by "wrapping" the end of the time-series data around to the beginning.

In time-series bootstrapping, there has also been research into varying the sizes of the time-series block for better theoretical properties in parameter estimation (e.g. see [47]), which could also be an interesting avenue of study for spatial methods.

B.2 Time-Series Cross-Validation

Much of the literature on CV model selection for dependent time-series data focuses on developing CV-based methods for bandwidth selection in nonparametric regression. Nonparametric regression methods are tuned by a bandwidth parameter h which controls how much the regression smooths the data. CV, particularly LOOCV, is commonly used to select an optimal bandwidth by simply estimating model error for different values of h and choosing the value of h with the lowest corresponding estimated error.[1]

While simple CV methods work well for bandwidth selection on independent data, [12] notes that for positively correlated observations, they choose bandwidths that are too small, and for negatively correlated observations, they choose bandwidths that are too large. To deal with these issues, [12] proposes *modified cross-validation* (MCV). For time-series observations $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots$ LOOCV is performed, except each training set $\mathcal{T}_i$ excludes not only the observation $(\mathbf{x_i}, y_i)$ in the validation set, but also the observations $(\mathbf{x_j}, y_j)$, for $j = i - l, \ldots, i + l$ for a given window width l. The excluded observations $(\mathbf{x_j}, y_j)$ act as a buffer region between the training and validation sets. [12] shows that as l increases, the effect of dependence on the selected bandwidth size decreases.

Variations of MCV can be found in [28] and [11], which introduce *one-sided cross-validation* (OSCV) and *far casting cross-validation* (FSCV), respectively. OSCV is identical to MCV, except the buffer region is constructed by omitting from the training set all of the points to the "left" or "right" of $(\mathbf{x_i}, y_i)$, i.e. all points $(\mathbf{x_j}, y_j)$ such that $j < i$ or $j > i$. FSCV simply extends MCV to more than one dimension, by using all points within a given n-dimensional distance d from $(\mathbf{x_i}, y_i)$ as buffer regions for each fold.

[9] takes a slightly different approach with *h-block cross-validation*. MCV, OSCV, and FSCV all select h by cross-validating nonparametric regressions with different values of h and choosing the one with the least error. In contrast, h-block CV cross-validates linear models. It selects an optimal value of h by using MCV-like buffer regions of size h, so that each linear model is trained on $N - 2h - 1$ observations and validated on a single observation. [48] introduces *hv-block cross-validation*, a natural extension of h-block CV

[1] For a more theoretical examination of the use of CV for bandwidth selection, see [27].

that uses an additional window of width v for validation-set points. Figure B.1 shows one fold each of h-block CV (top) and hv-block CV (bottom) for thirteen observations, where the validation set is red, the training set green, and the buffer black.

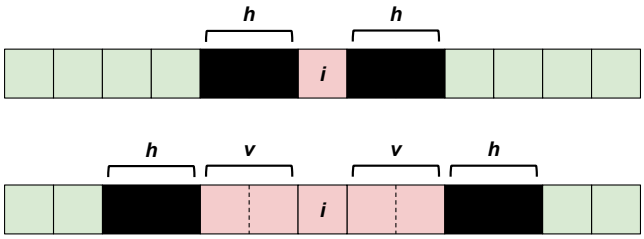


Figure B.1: One fold of h-block and hv-block CV

Importantly, [48] considers whether h-block CV or hv-block CV satisfy the three conditions for consistency laid out in [52]: $\lim_{N\to\infty} N_t = \infty$, $\lim_{N\to\infty} \frac{N_t}{N} = 0$, and $\lim_{N\to\infty} \frac{N_v}{N} = 1$, where N_t is the size of each fold's training set and N_v is the size of each fold's validation set. It shows that while h-block CV is not consistent (since $\lim_{N\to\infty} \frac{N_v}{N} \neq 1$), hv-block CV is, so long as h and v are chosen to satisfy the given conditions. We note that Figure B.1 shows how hv-block CV uses more observations for validation, which matches the suggestions of Shao's papers on the consistency of CV and the bootstrap (see [52], [53]).

Finally, [12] also mentions *partitioned CV* (PCV), first introduced in [42]. PCV is K-fold CV, except instead of randomly assigning observations to K folds, the observations are assigned by systematically taking every Kth observation in time for each fold. [12] shows that increasing K decreases the effect of dependence on the selected bandwidth size, but that PCV actually turns out to be "too effective at removing dependence." Asymptotically, PCV with large K leads to the selection of the optimal bandwidth for data with no dependences, which differs from the optimal for dependent data.

In the context of this thesis, we can reframe these results for bandwidth selection in terms of spatial resampling for model selection. The time-series CV methods discussed here are similar in some ways to the spatial CV methods described in Chapter 3. MCV and FSCV are essentially SLOO in different dimensions. hv-block CV is more similar to buffered grid CV, since it considers neighborhoods for both the validation set and the buffer. PCV

is similar to naive K-fold CV on spatial data, where folds are chosen to be uniformly spread across the study region (which is what would be expected from random sampling). Thus, it is encouraging that [12] shows that MCV has good theoretical and empirical properties, [11] shows FSCV outperforming LOOCV and other methods in simulation studies, and that [48] can prove the consistency of hv-block CV, since, at least intuitively, it suggests that the spatial CV methods described in Chapter 3 may have similar properties. It is also encouraging that PCV does not appropriately account for dependences in the data, since we developed the CV and bootstrap methods in Chapter 3 to account for the shortcomings of methods like naive K-fold CV.

Appendix C

Chapter 5 Simulation Details

This appendix covers the details behind the simulations in Chapter 5.

C.1 Model Assessment and Selection Simulations

C.1.1 Generating Spatially Autocorrelated Variables

We generate spatially autocorrelated variables using multivariate normal random variables: given N points, we want to compute a N-dimensional multivariate normal random variable for each spatially autocorrelated variable (in our case, `X1`, `X2`, `X3`, and `eps`). To do so, we need a N-dimensional mean vector and a $N \times N$ covariance matrix.

For the mean vector of each variable, we just use the zero vector. Computing the $N \times N$ covariance matrix requires more work. First, we compute the $N \times N$ pairwise (Euclidean) distance matrix for the points. Then, we apply a continuous distance-decay function $f : [0, \infty] \to (0, 1)$ to the distance matrix. f describes how correlation between points decreases as the distance between points increases. Points that are very close to each other should have correlation ≈ 1, and points that are very far from each other should have correlation ≈ 0, with the function monotonically decreasing as $x \to \infty$. In our simulations, we use the function

$$f(x) = 1 - \frac{x}{x + \alpha}, \text{ for } \alpha = 10.$$

There are many valid choices for f. We make an ad hoc choice of this functional form and $\alpha = 10$ because the resulting function "looks like" how we might expect spatial autocorrela-

tion to decay over space. It could be interesting to try other values of α[1] or forms of f. One important practical note is that the resulting matrix should not be too sparse, i.e. f should not equal zero for too many of the N^2 distances. This is because the computation of the multivariate normal random variable inverts the covariance matrix, and sparser matrices are more likely to have determinant zero.

Finally, we scale the distance matrix by a given population variance σ^2. This is because the diagonal of the covariance matrix should be the population variance of the points. For our simulation, we choose $\sigma = 0.5$, again in an ad hoc manner. This produces our final covariance matrix, which we use to randomly generate our spatially autocorrelated variables.

C.1.2 Choosing Assessment/Selection Methods

In order to decide on using training error, K-fold CV error, LOO bootstrap error, buffered grid CV error, and moving circle bootstrap error for our model assessment and selection simulations, we first ran a series of other simulations.

We compare nine model assessment methods:[2]

Nonspatial Resampling Methods	**Spatial CV Methods**	**Spatial Bootstrap Methods**
Training Error	Grid CV	Fixed Tile Bootstrap
K-fold CV Error	Buffered Grid CV	Moving Circle Bootstrap
LOO Bootstrap	SLOO	Moving Tile Bootstrap (with toroidal wrapping)

For each parametrized assessment method (e.g. K-fold CV with parameter K), we compare the performances of the assessment method with different values of the parameter using the same model assessment simulation as we use in Chapter 5. The results of these simulations are shown in Figures C.1 and C.2. For Figure C.1, the parameter values for each method are plotted along the x-axis, and the y-axis shows the average (over all 100 samples) absolute value of the difference between the true error and the estimated error of the model (using the given parameter value). Figure C.2 considers the moving tile and circle bootstraps, which have two tuning parameters. The parameter values are plotted

[1] As α increases from 1 to ∞, the function decays more slowly.

[2] Future extensions may want to consider SKCV as well.

along the axes and the average absolute difference between true error and estimated error is visualized as the color of the tiles (darker tiles represent better estimates).

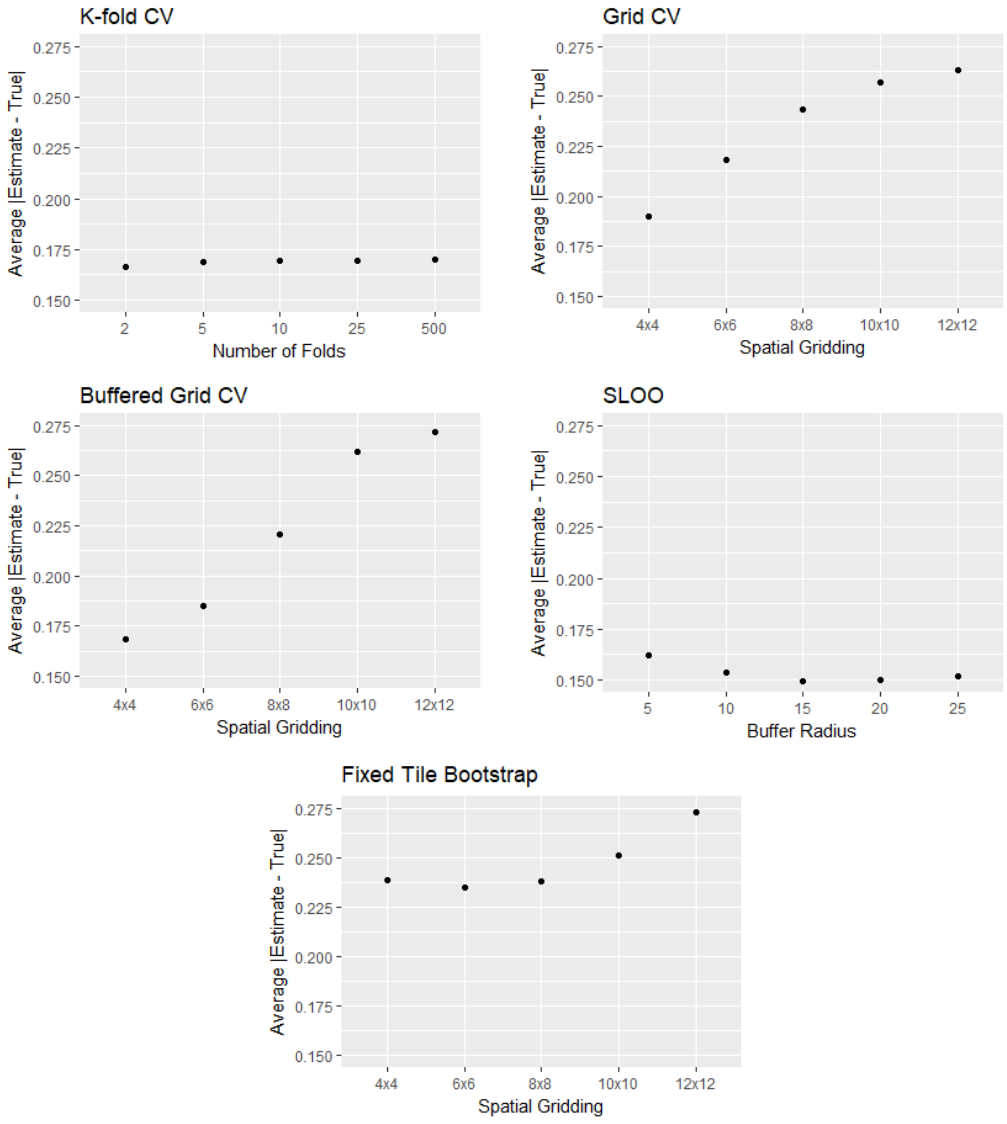


Figure C.1: Assessment performance across different parameter values

We draw two main conclusions from these simulations. First, while parameter values do not significantly impact K-fold CV's performance on the data, they have large effects on most of the spatial methods. Second, we note that training on less data generally improves estimation accuracy. Though it is not shown in Figures C.1 and C.2, all of the error-estimation methods underestimate true error. We suspect that training on less data may

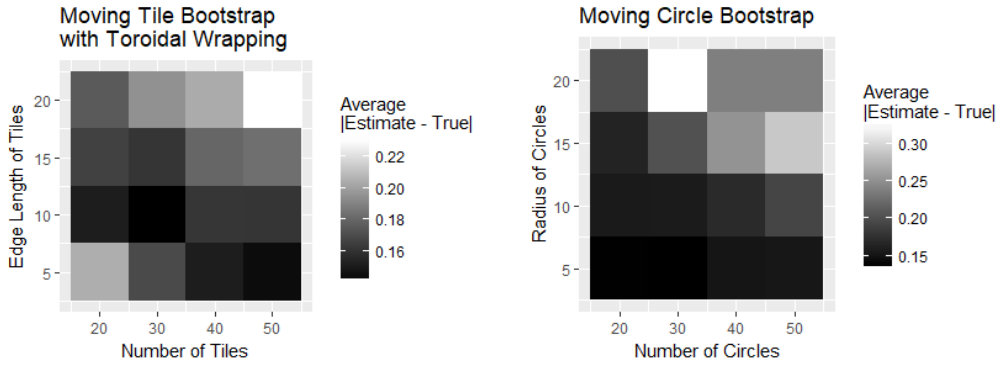


Figure C.2: Assessment performance across different parameter values

lead to underfitting (i.e. greater error on the training data), which leads to greater error estimates (i.e. error estimates closer to true error). Thus, we consider these results with a grain of salt.

We choose our six methods based on these simulations. For "baseline" nonspatial comparisons, we choose to use training error, K-fold CV, and the LOO bootstrap. We let $K = 10$ because it is the canonical approach recommended in the literature (e.g. [34]). In any case, K does not seem to have a significant effect on performance. We use SLOO (with buffer radius 15) because of its good performance on our simulations. Noting that grid CV and buffered grid CV have similar performances, we choose to use buffered grid CV (on a 4x4 grid) because of the good intuitive properties of the buffer. Finally, we note that of the bootstrap methods, the moving circle bootstrap seems to have the best performance, so we use a moving circle bootstrap with 30 circles of radius 5.

C.1.3 Functions Used

The model assessment function uses the function $f(x, y, z) = 2\sin(\pi x) + 2y + 4 * \mathbb{1}_{z>0}(z)$, where $\mathbb{1}_{z>0}(z)$ is the indicator function that is 1 if $z > 0$ and 0 otherwise. The model selection function uses the function $f(x, y) = 2\sin(\pi x) + 2y$. Admittedly, these functions are chosen in a relatively arbitrary manner. In general, the goal is to make the signal roughly linear in some sense, but complex enough that the learning curve for a linear model will not achieve a minimum with too few training observations (see Section 5.3).

C.2 Real-World Data Study

C.2.1 Description of Data

This section briefly describes the data collected by Albert Y. Kim and David Allen. The dataset contains infomation about trees on the University of Michigan's Edwin S. George Reserve. At two points in time (2008 and 2014), a census of all of the trees in a given area of land was taken. For each tree, information about the tree's species and size (measured by the tree's diameter at breast height, or dbh) was recorded at both times.

The dataset contains the following variables:

- `x`, `y`: the spatial x- and y-coordinates of the tree
- `species`: the species of the tree
- `dbh`: the tree's diameter at breast height, in 2008
- `ncomp`: the number of competitor trees[3]
- `biomass_comp`: the total biomass of the competitor trees
- `growth`: the (positive) difference between the tree's dbh in 2014 and in 2008

For the sake of simplicity, the dataset that we use in Chapter 5 includes only trees that are alive in both 2008 and 2014. Also, it accounts for edge effects on the `ncomp` and `biomass_comp` variables using a 7.5-meter buffer region around the edges of the study region. For trees in the buffer region, information is collected and incorporated into the values of `ncomp` and `biomass_comp` for trees not in the buffer region, but the trees themselves are not included in the dataset.

C.2.2 Choosing Assessment Methods

For the study, we use the six assessment methods used in the model assessment and selection simulations, with the exception of SLOO. This is because the dataset has over 20,000 observations, with 2-4,000 observations per strip. Though it is not a massive dataset, its size made it computationally difficult to run SLOO without running into memory or runtime barriers, so we chose to not use SLOO. We assess the model $\texttt{growth} \sim \texttt{species} + \texttt{dbh} + \texttt{ncomp} + \texttt{biomass\_comp}$.

[3]Defined to be trees within 7.5 meters of the given tree

For K-fold CV, we let $K = 10$, for the same reasons as we gave in the model assessment study. For the buffered grid CV, we divide strips 1-5 into 16 square regions, and the smaller strip 6 into 12 square regions. Notably, the different shapes of the regions lead to different amounts of data being excluded by the CV buffers, but we do not explore the effects of this any further. Finally, for the moving circle bootstrap, we take 60 circles of radius 15 for each bootstrap. We choose these values because we found them to produce training sets of similar sizes as the LOO bootstrap, and felt that made for a more intuitive comparison between the two methods.

C.2.3 Assessment Study Details

Because there are only six sets of data (rather than 100, like in our model assessment simulation), we can visualize each model's performance on each set. Figure C.3 plots, for the linear model trained on each strip, the model's error on its training strip (training error, indicated by triangular points) as well as on each of the other five strips. We see that strips

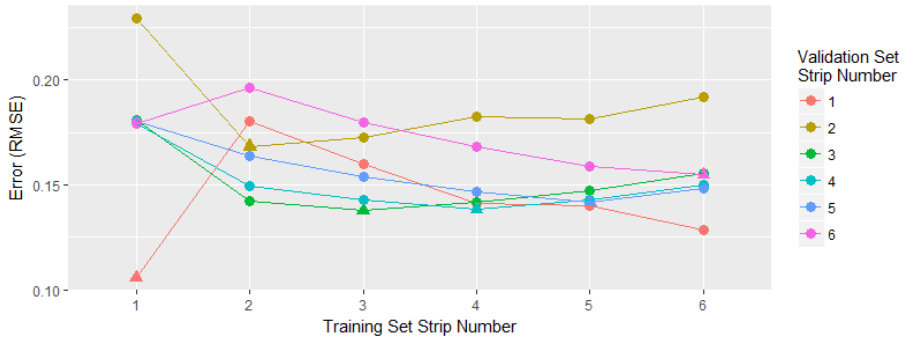


Figure C.3: Model performances on six strips

2 and 6 are particularly difficult to predict on, and that models trained on strip 1 do not perform well on other strips.

Figure C.4 shows the other four facets of Figure 5.3 that were excluded in Section 5.2. Results are mixed. For some strips (e.g. 1 and 2), the moving circle bootstrap has good performance, while on others (e.g. 3) it clearly performs poorly. Buffered grid CV seems to have generally good performance, though it does perform relatively poorly on strip 6.[4]

[4]Notably, strip 6 has a squarer shape than the other strips, so buffered grid CV will include more data in its buffers, on average, which may affect its performance.

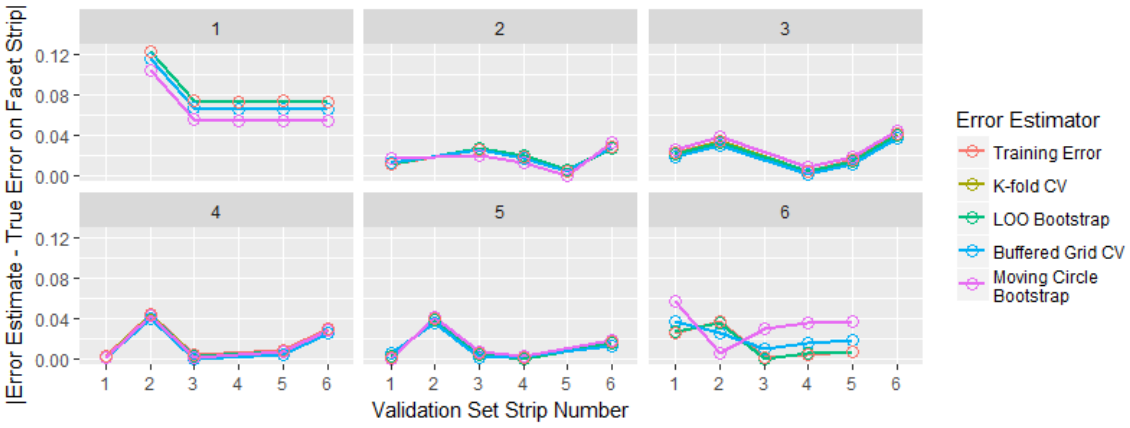


Figure C.4: Performances of assessment methods for all strips

We draw the general conclusion that training error, K-fold CV, and the LOO bootstrap have similar performance on the real dataset, and that spatial methods usually (but don't always) outperform nonspatial methods at assessment.

Appendix D

Code Appendix

This appendix contains (almost) all of the code used in this thesis. For conciseness, we do our best to note places where similar code is reused. The electronic versions of this code contain more details (including how files depend on each other, some exploratory data analysis to confirm that results are correct, etc.). For our purposes here, we try to present the code in as simple a manner as possible. All analyses are conducted within the R "tidyverse" (`require(tidyverse)`), see https://www.tidyverse.org/.

D.1 CV and Bootstrap Procedures

For the sake of consistency and clarity, we built wrapper functions for all of the model-assessment procedures we used. Each function inputs a dataframe, the parameters for the assessment method, and a model formula. Each function outputs a single (RMSE) estimate of the model error.

```
require(tidyverse)
require(sperrorest)   # for spatial methods
```

K-fold CV

```
cv_lm <- function(df, nfolds, model_formula) {
  cv_errors <- vector(mode="numeric", length=nfolds)

  # Shuffle data and partition folds
  temp <- df[sample(nrow(df)),]
  folds <- cut(seq(1,nrow(temp)),
               breaks=nfolds,
```

```
                labels=FALSE)
  for(i in 1:nfolds){
    # Split training/testing sets
    test_index <- which(folds==i, arr.ind=TRUE)
    test <- temp[test_index, ]
    train <- temp[-test_index, ]
    # Fit lm
    fitted_model <- lm(model_formula, data = train)
    # Store error
    diffs <- predict(fitted_model, test) - test$growth
    cv_errors[i] <- mean(diffs^2)
  }
  return(sqrt(mean(cv_errors)))
}
```

Flipped K-fold CV

```
cv_lm_flip <- function(df, nfolds, cp, model_formula) {
  # Empty dataframe with 1 column per observation in df
  obs_errors <- matrix(nrow=0, ncol=nrow(df))

  # Shuffle data and partition folds
  temp <- df[sample(nrow(df)),]
  folds <- cut(seq(1,nrow(temp)),
               breaks=nfolds,
               labels=FALSE)
  for(i in 1:nfolds){
    # Split training/testing sets
    train_index <- which(folds==i, arr.ind=TRUE)
    train <- temp[train_index, ]
    test <- temp[-train_index, ]
    # Fit lm
    fitted_model <- lm(model_formula, data = train)
    # Store error for each observation (0 for obs in train)
    err_vec <- vector(mode="numeric", length=nrow(df))
    diffs <- predict(fitted_model, test) - test$growth
    err_vec[-train_index] <- diffs
    obs_errors <- rbind(obs_errors, err_vec)
  }
  # Average error for each observation, then rmse
  errs <- (rowSums(t(obs_errors))/(nfolds-1))^2
  return(sqrt(mean(errs)))
}
```

Buffered Grid CV

```r
cv_grid_buffer_lm <- function(df, nrow, ncol, model_formula) {
  # Use sperrorest to partition data into spatial folds
  nfolds_spat <- nrow*ncol
  partition <- partition_tiles(
    as.data.frame(df),
    nsplit = c(ncol,nrow),    # nrow x ncol grid
    reassign = FALSE)   # Don't reassign observations in small folds

  # Hacky method to number which fold each observation is in
  all_folds <- partition$`1`
  count = 1
  fold_df <- df %>%
    select(x, y)
  for (fold in all_folds){
    varname <- paste("fold", count, sep="")
    fold_df <- fold_df %>%
      mutate(!!varname := count*(row_number() %in% fold$test))
    count <- count + 1
  }
  folds <- rowSums(fold_df[,c(3:ncol(fold_df))])
  df_spatial <- df %>%
    mutate(fold = folds)

  # Find fold neighbors of each fold (again, hacky)
  fold_neighbors <- vector(mode="list", length=nfolds_spat)
  for (i in 1:nfolds_spat) {
    list1 <- c(i, i-nrow, i+nrow)
    list2 <- switch(as.character(i %% nrow),
                    "0" = c(i-1, i-1-nrow, i-1+nrow),
                    "1" = c(i+1, i+1-nrow, i+1+nrow),
                    c(i-1, i-1-nrow, i-1+nrow, i+1, i+1-nrow, i+1+nrow))
    fold_neighbors[[i]] <- c(list1, list2)
  }

  # Compute buffered grid CV error
  results_list <- vector(mode="numeric", length=nfolds_spat)
  num_empty_folds <- 0
  for (j in 1:nfolds_spat) {
    # Split training/testing sets
    train <- df_spatial %>% filter(!(fold %in% fold_neighbors[[j]]))
    test <- df_spatial %>% filter(fold == j)
    # Account for empty test sets in mean
    if (dim(test)[1] == 0) {
      num_empty_folds <- num_empty_folds + 1
      next
    }
    # Fit lm & make predictions
    model_lm <- lm(model_formula, data=train)
    model_preds <- predict(model_lm, test)
    # Store results
    results_list[[j]] <- rmse(model_preds, test$growth)
```

```
  }
  return(sum(unlist(results_list)) / (nfolds_spat-num_empty_folds))
}
```

Flipped Buffered Grid CV

```
cv_grid_buffer_lm_flip <- function(df, nrow, ncol, model_formula) {
  # Use sperrorest to partition data into spatial folds
  nfolds_spat <- nrow*ncol
  partition <- partition_tiles(
    as.data.frame(df),    # See testing_sperrorest_package.Rmd
    nsplit = c(ncol,nrow),    # nrow x ncol grid
    reassign = FALSE)    # Don't reassign observations in small folds

  # Hacky method to number which fold each observation is in
  all_folds <- partition$`1`
  count = 1
  fold_df <- df %>%
    select(x, y)
  for (fold in all_folds){
    varname <- paste("fold", count, sep="")
    fold_df <- fold_df %>%
      mutate(!!varname := count*(row_number() %in% fold$test))
    count <- count + 1
  }
  folds <- rowSums(fold_df[,c(3:ncol(fold_df))])

  # Find fold neighbors of each fold (again, hacky)
  fold_neighbors <- vector(mode="list", length=nfolds_spat)
  for (i in 1:nfolds_spat) {
    list1 <- c(i, i-nrow, i+nrow)
    list2 <- switch(as.character(i %% nrow),
                    "0" = c(i-1, i-1-nrow, i-1+nrow),
                    "1" = c(i+1, i+1-nrow, i+1+nrow),
                    c(i-1, i-1-nrow, i-1+nrow, i+1, i+1-nrow, i+1+nrow))
    fold_neighbors[[i]] <- c(list1, list2)
  }

  # Compute flipped buffered grid CV error
  # TODO: include check for empty training sets
  obs_errors <- matrix(nrow=0, ncol=nrow(df))
  for (j in 1:nfolds_spat) {
    train_index <- which(folds == j, arr.ind=TRUE)
    test_index <- which(!(folds %in% fold_neighbors[[j]]), arr.ind=TRUE)
    train <- df[train_index,]
    test <- df[test_index,]
    # Train model & make predictions
    model_lm <- lm(model_formula, data=train)
    # Store error for each observation (0 for obs in train)
```

```
    err_vec <- vector(mode="numeric", length=nrow(df))
    diffs <- predict(model_lm, test) - test$growth
    err_vec[test_index] <- diffs
    obs_errors <- rbind(obs_errors, err_vec)
  }
  # Average errors for each observation, accounting for fact that
  # each observation may have a different number of estimated errors
  errs <- vector(mode="numeric", length=nrow(df))
  for (k in 1:nrow(df)) {
    temp <- obs_errors[,k]
    num_zero <- sum(temp == 0)
    errs[k] <- (sum(temp)/(nfolds_spat-num_zero))^2
  }
  return(sqrt(mean(errs)))
}
```

Unbuffered Grid CV

```
cv_grid_nobuff_lm <- function(df, nrow, ncol, model_formula) {
  model <- sperrorest(
    model_formula,
    data=df,
    coords=c("x", "y"),   # variables that contain x/y coordinates
    model_fun=glm,
    pred_fun=predict,
    smp_fun=partition_tiles,   # spatial cv by rectangular grids
    smp_args = list(repetition = 1,
                    nsplit = c(nrow, ncol),
                    reassign = FALSE,
                    seed1 = 123),
    progress=FALSE)
  return(summary(model$error_fold)["test.rmse",]$mean)
}
```

SLOO CV

```
cv_SLOO_lm <- function(df, buffer, model_formula) {
  model <- sperrorest(
    model_formula,
    data=df,
    coords=c("x", "y"),
    model_fun=glm,
    pred_fun=predict,
    smp_fun=partition_loo,   # LOO CV
    smp_args = list(repetition = 1,
                    buffer = buffer,   # Buffer for SLOO
```

```
                    seed1 = 123),
    progress=FALSE)
  return(summary(model$error_rep)["test_rmse",]$mean)
}
```

LOO Bootstrap

```
boot_LOO_lm <- function(df, nboot=500, model_formula) {
  N <- nrow(df)
  boot_errors <- vector(mode="numeric", length=nboot)
  for (i in 1:nboot) {
    # Partition training/validation sets
    samp_indices <- sample(1:N, N, replace=TRUE)
    train <- df[unique(samp_indices),]
    test <- df[-samp_indices,]
    # Fit lm
    fitted_model <- lm(model_formula, data = train)
    # Return error
    diffs <- predict(fitted_model, test) - test$growth
    boot_errors[i] <- mean(diffs^2)
  }
  return(sqrt(mean(boot_errors)))
}
```

Fixed Tile Bootstrap

```
boot_fixedtile_lm <- function(df, nrow, ncol, model_formula) {
  model <- sperrorest(
    model_formula,
    data=as.data.frame(df),
    coords=c("x", "y"),
    model_fun=glm,
    pred_fun=predict,
    smp_fun=represampling_tile_bootstrap,    # fixed tile bootstrap
    smp_args = list(repetition = 1,
                    nsplit = c(nrow, ncol),
                    reassign = FALSE,
                    seed1 = 44),
    progress=FALSE)
  return(summary(model$error_fold)["test.rmse",]$mean)
}
```

Moving Tile Bootstrap with Toroidal Wrapping

```
boot_movingtiletw_lm <- function(df,max_x,max_y,size,nboot,model_formula) {
  # Helper function that randomly samples a tile of
  # width `size`, with toroidal wrapping
  sample_tile_tw <- function(df, size=size) {
    # Randomly generate center of tile
    x_coord <- runif(1, 0, max_x)
    y_coord <- runif(1, 0, max_y)
    # Sample tile, with toroidal wrapping
    temp <- as.data.frame(df) %>%
      mutate(index = row_number()) %>%
      filter(
        x > x_coord-(size/2) | x < x_coord-max_x+(size/2),
        y > y_coord-(size/2) | y < y_coord-max_y+(size/2),
        x < x_coord+(size/2) | x > x_coord+max_x-(size/2),
        y < y_coord+(size/2) | y > y_coord+max_y-(size/2))
    return(temp$index)
  }

  # Sample nboot tiles
  train_indices <- c()
  for (i in 1:nboot) {
    train_indices <- c(train_indices, sample_tile_tw(df, size))
  }
  train_indices <- unique(train_indices)

  # Partition training/validation sets
  train <- df[train_indices,]
  test <- df[-train_indices,]
  # Fit lm
  fitted_model <- lm(model_formula, data=train)
  # Return error
  diffs <- predict(fitted_model, test) - test$growth
  return(sqrt(mean(diffs^2)))
}
```

Moving Circle Bootstrap

```
boot_movingcircle_lm <- function(df, radius, nboot, model_formula) {
  # Get points for training set of moving circle bootstrap
  parti <- represampling_disc_bootstrap(
    as.data.frame(df),
    radius=radius,
    nboot=nboot)
  samp_indices <- parti[[1]][1]$`1`$train

  # Partition training/validation sets
  train <- df[unique(samp_indices),]
  test <- df[-samp_indices,]
  # Fit lm
```

```
  fitted_model <- lm(model_formula, data=train)
  # Return error
  diffs <- predict(fitted_model, test) - test$growth
  return(sqrt(mean(diffs^2)))
}
```

D.2 Model Assessment and Selection Simulations

`generate_sample` randomly generates one sample for our simulations in Chapter 5. The `mvnfast` package is used to generate the multivariate normal because in practice it was far faster than the traditional `mvrnorm` package.

```
require(mvnfast)

generate_sample <- function(f, max_x=100, max_y=100, npoints=500, sigma=0.5) {
  # Generate x/y coordinates, uniformly across space
  x <- runif(npoints, 0, max_x)
  y <- runif(npoints, 0, max_y)
  df <- data.frame(x,y)

  # Generate spatially correlated X1, X2, X3, noise
  mat <- as.matrix(proxy::dist(df))   # distance matrix
  sigma_mat <- 1 - mat/(mat+10)   # function to simulate distance decay
  sigma_mat <- (sigma^2)*sigma_mat   # diagonals equal variance
  # Generate nonspatial normal noise
  noise <- rnorm(npoints, 0, sigma)

  vars <- rmvn(
    n=4,   # number of variables to generate
    mu=rep(0, npoints),   # vector of variable means
    sigma=sigma_mat,
    ncores=4)
  df <- cbind(df, vars[1,], vars[2,], vars[3,], vars[4,])
  colnames(df) <- c("x","y","X1","X2","X3","eps")
  df <- df %>%
    mutate(growth = f(X1,X2,X3) + eps + noise)
  return(df)
}
```

Choosing Assessment Methods

The code in this section covers the results of Section C.1.2. We only show how to create the top-left plot in Figure C.1 (for K-fold CV). The code to generate the other figures is

essentially the same. The only required modifications are to `param_vec` (to include the parameters of interest) and the CV method used to produce `err_vec`.

The simulation for K-fold CV used random seed 5, grid CV 10, buffered grid CV 10, SLOO CV 15, fixed tile bootstrap 20, moving tile bootstrap with toroidal wrapping 25, and moving circle bootstrap 30.

```
# Define parameters for simulation #
set.seed(5)
NUMSAMP <- 100   # Number of simulation samples to generate
NPOINTS <- 500   # Number of points per sample
# Boundaries of data to use
MAX_X <- 100
MAX_Y <- 100

param_vec <- c(2,5,10,25,500)   # Parameters to check

# Define true model
f <- function(x, y, z){
  b1 <- 2
  b2 <- 2
  b3 <- 4
  return(b1*sin(pi*x) + b2*(y) + b3*(z>0))
}
model_formula <- as.formula("growth ~ X1 + X2 + X3")

# Generate random samples
all_samps <- vector(mode = "list", length = NUMSAMP)
all_samps <- lapply(all_samps, function(x) {generate_sample(f)})

# Compute true error and error estimates for each sample
err_df <- data.frame()
for (i in 1:NUMSAMP) {
  # Get sample i
  df_train <- all_samps[[i]]

  # Fit lm to df_train
  model_lm <- lm(model_formula, data=df_train)
  # Compute test errors on all other training sets
  get_err <- function(df) {
    rmse(predict(model_lm, df), df$growth)
  }
  errs_test <- sapply(all_samps, get_err)
  # Compute true error (need to remove error for df_train)
  err_true <- (sum(errs_test)-errs_test[i]) / (NUMSAMP-1)

  err_vec <- lapply(param_vec, function(x){
    cv_lm(df_train, x, model_formula=model_formula)
  })
```

```
  # Output results
  err_df <- rbind(err_df, c(i, err_true, unlist(err_vec)))
}
colnames(err_df) <- c("set_number", "err_true",
                      sapply(param_vec, function(x) {
                        paste("err", x, sep="")
                      }))
```

The code below generates the top-left plot of Figure C.1.

```
err_df %>%
  select(-err3) %>%
  summarize(
    `2` = mean(abs(err2-err_true)),
    `5` = mean(abs(err5-err_true)),
    `10` = mean(abs(err10-err_true)),
    `25` = mean(abs(err25-err_true)),
    `500` = mean(abs(err500-err_true))
  ) %>%
  gather(err, val, `2`:`500`) %>%
  ggplot(aes(x=as.factor(as.numeric(err)), y=val)) +
  geom_point() +
  coord_cartesian(ylim=c(.15, .275)) +
  labs(
    title="K-fold CV",
    y="Average |Estimate - True|",
    x="Number of Folds"
  )
```

Model Assessment Simulation

As discussed in the thesis, this is the same code as is used to choose assessment methods. We display this code in its entirety here, but for the sake of conciseness will be truncating similar code moving forward.

```
# Define parameters for simulation
set.seed(500)
MAX_X <- 100   # Boundaries of data to use
MAX_Y <- 100
NUMSAMP <- 100   # Number of simulation samples to generate
NPOINTS <- 500   # Number of points per sample

NFOLDS <- 10   # Number of folds for K-fold cv
NROW <- 4   # Number of rows for grid CV
NCOL <- 4   # Number of columns for grid CV
RADIUS_SLOO <- 15   # Size of buffer for SLOO
```

```
RADIUS <- 5   # Size of each moving circle bootstrap sample
NBOOT <- 30   # Number of moving circle bootstrap samples to take

# Define true model
f <- function(x, y, z){
  b1 <- 2
  b2 <- 2
  b3 <- 4
  return(b1*sin(pi*x) + b2*(y) + b3*(z>0))
}
model_formula <- as.formula("growth ~ X1 + X2 + X3")

# Generate random samples
all_samps <- vector(mode = "list", length = NUMSAMP)
all_samps <- lapply(all_samps, function(x) {generate_sample(f)})

# Compute true error and error estimates (1-5) for each sample
err_df <- data.frame()
for (i in 1:NUMSAMP) {
  # Get sample i
  df_train <- all_samps[[i]]

  # Fit lm to df_train
  model_lm <- lm(model_formula, data=df_train)

  # Compute test errors on all other training sets
  get_err <- function(df) {
    rmse(predict(model_lm, df), df$growth)
  }
  errs_test <- unlist(lapply(all_samps, get_err))
  # Compute true error (need to remove error for df_train)
  err_true <- (sum(errs_test)-errs_test[i]) / (NUMSAMP-1)

  # Compute training error (1)
  err_train <- rmse(predict(model_lm, df_train), df_train$growth)

  # Compute K-fold CV error (2)
  err_cv <- cv_lm(df_train, NFOLDS, model_formula=model_formula)

  # Compute LOO bootstrap error (3)
  err_LOO <- boot_LOO_lm(df_train, model_formula=model_formula)

  # Compute buffered grid CV error (4)
  err_grid_buffer <- cv_grid_buffer_lm(df_train, NROW, NCOL,
    model_formula=model_formula)

  # Compute SLOO CV error (5)
  err_SLOO <- cv_SLOO_lm(df_train, RADIUS_SLOO, model_formula=model_formula)

  # Compute moving circle bootstrap error (6)
  err_moving <- boot_movingcircle_lm(df_train, RADIUS, NBOOT,
```

```
    model_formula=model_formula)

  # Output results
  err_df <- rbind(err_df, c(i, err_true, err_train, err_cv, err_LOO,
    err_grid_buffer, err_SLOO, err_moving))
}
colnames(err_df) <- c("set_number", "err_true", "err_train",
                      "err_cv", "err_LOO", "err_grid_buffer",
                      "err_SLOO", "err_moving")
```

The code below generates Table 5.1.

```
require(kable)
require(kableExtra)

err_df %>%
  rename(
    `Training Error` = err_train,
    `K-fold CV` = err_cv,
    `LOO Bootstrap` = err_LOO,
    `Buffered Grid CV` = err_grid_buffer,
    `SLOO CV` = err_SLOO,
    `Moving Circle Bootstrap` = err_moving
  ) %>%
  gather(
    `Assessment Method`, est_val,
    `Training Error`:`Moving Circle Bootstrap`) %>%
  group_by(`Assessment Method`) %>%
  summarize(`Average Error Estimate` = mean(est_val)) %>%
  arrange(factor(`Average Error Estimate`)) %>%
  kable(
    format = "latex",
    digits = 3,
    booktabs = T)
```

The code below generates Figure 5.1.

```
err_df %>%
  rename(
    `Training Error` = err_train,
    `K-fold CV` = err_cv,
    `LOO Bootstrap` = err_LOO,
    `Buffered Grid CV` = err_grid_buffer,
    `SLOO CV` = err_SLOO,
    `Moving Circle Bootstrap` = err_moving) %>%
  gather(err_est, est_val, `Training Error`:`Moving Circle Bootstrap`) %>%
  mutate(err_est = factor(err_est, levels=c("Training Error",
    "K-fold CV", "LOO Bootstrap", "Buffered Grid CV",
    "SLOO CV","Moving Circle Bootstrap"))) %>%
```

```
  ggplot((aes(x=est_val-err_true))) +
  geom_density(aes(color=err_est, linetype=err_est)) +
  scale_linetype_manual(
    name="Error Estimation Method",
    values=c("solid", "dashed", "dotted", "solid", "dashed", "dotted")) +
  scale_color_manual(
    name="Error Estimation Method",
    values=c("black", "black", "black", "red", "red", "red")) +
  geom_vline(xintercept=0) +
  coord_cartesian(xlim=c(-0.5, 0.3)) +
  labs(
    y="Density",
    x="Error Estimate - True Error",
    color="Error Estimation Method"
  )
```

Model Selection Simulation

The code here is similar to the assessment code. Regardless, we display it in its entirety here. Note that it produces `sel_df`, which contains only the models selected by each assessment method, and `errs_df`, which contains the errors estimated for each model. We do not end up using `errs_df` in the thesis, though it was studied.

```
# Define parameters for simulation
set.seed(600)
MAX_X <- 100   # Boundaries of data to use
MAX_Y <- 100
NUMSAMP <- 100   # Number of simulation samples to generate
NPOINTS <- 500   # Number of points per sample

NFOLDS <- 10   # Number of folds for K-fold cv
NROW <- 4   # Number of rows for grid CV
NCOL <- 4   # Number of columns for grid CV
RADIUS_SLOO <- 15   # Size of buffer for SLOO
RADIUS <- 5   # Size of each moving circle bootstrap sample
NBOOT <- 30   # Number of moving circle bootstrap samples to take

# Define true model
f <- function(x, y, z){
  b1 <- 2
  b2 <- 2
  return(b1*sin(pi*x) + b2*(y))
}

# Generate random samples
all_samps <- vector(mode = "list", length = NUMSAMP)
all_samps <- lapply(all_samps, function(x) {generate_sample(f)})
```

```
# Compute true error and error estimates (1-5) for each sample, model
# --> Select model with least error
sel_df <- data.frame()
errs_df <- data.frame()
for (i in 1:NUMSAMP) {
  # Get sample i
  df_train <- all_samps[[i]]
  model_formulas <- list(
    as.formula("growth ~ X1"),
    as.formula("growth ~ X2"),
    as.formula("growth ~ X1 + X2"),
    as.formula("growth ~ X1 + X2 + X3"))

  # Compute training error (1) for each model
  get_err_train <- function(formula) {
    model_lm <- lm(formula, data=df_train)
    return(rmse(predict(model_lm, df_train), df_train$growth))
  }
  errs_train <- lapply(model_formulas, get_err_train)
  # Select model with least training error
  sel_train <- which.min(errs_train)

  # Compute K-fold CV error (2) for each model
  get_err_cv <- function(formula) {
    return(cv_lm(df_train, NFOLDS, model_formula=formula))
  }
  errs_cv <- lapply(model_formulas, get_err_cv)
  # Select model with least K-fold CV error
  sel_cv <- which.min(errs_cv)

  # Compute LOO bootstrap error (3) for each model
  get_err_LOO <- function(formula) {
    return(boot_LOO_lm(df_train, model_formula=formula))
  }
  errs_LOO <- lapply(model_formulas, get_err_LOO)
  # Select model with least LOO bootstrap error
  sel_LOO <- which.min(errs_LOO)

  # Compute buffered grid CV error (4) for each model
  get_err_grid_buffer <- function(formula) {
    return(cv_grid_buffer_lm(df_train, NROW, NCOL, model_formula=formula))
  }
  errs_grid_buffer <- lapply(model_formulas, get_err_grid_buffer)
  # Select model with least buffered grid CV error
  sel_grid_buffer <- which.min(errs_grid_buffer)

  # Compute spatial LOO bootstrap error (5) for each model
  get_err_SLOO <- function(formula) {
    return(cv_SLOO_lm(df_train, RADIUS_SLOO, model_formula=formula))
  }
```

```
  errs_SLOO <- lapply(model_formulas, get_err_SLOO)
  # Select model with least LOO bootstrap error
  sel_SLOO <- which.min(errs_SLOO)

  # Compute moving circle bootstrap error (6) for each model
  get_err_moving <- function(formula) {
    return(boot_movingcircle_lm(df_train, RADIUS, NBOOT,
      model_formula=formula))
  }
  errs_moving <- lapply(model_formulas, get_err_moving)
  # Select model with least moving circle bootstrap error
  sel_moving <- which.min(errs_moving)

  # Output results of both selection and assessment
  sel_df <- rbind(sel_df, c(i, sel_train, sel_cv, sel_LOO, sel_grid_buffer,
    sel_SLOO, sel_moving))
  errs_df <- rbind(errs_df, data.frame(
    rep(i, 6),
    c("train", "cv", "LOO", "grid_buffer", "SLOO", "moving"),
    t(matrix(c(errs_train, errs_cv, errs_LOO, errs_grid_buffer,
      errs_SLOO, errs_moving), nrow=4, ncol=6))))
}
colnames(sel_df) <- c("set_number", "sel_train", "sel_cv", "sel_LOO",
                      "sel_grid_buffer", "sel_SLOO", "sel_moving")
colnames(errs_df) <- c("set_number", "method", "err1", "err2", "err3", "err4")
```

The code below generates Table 5.2.

```
order_vec <- c("sel_train", "sel_cv", "sel_LOO",
  "sel_grid_buffer", "sel_SLOO", "sel_moving")
sel_df %>%
  gather(`Selection Method`, model_selected, sel_train:sel_moving) %>%
  group_by(`Selection Method`) %>%
  summarize(
    `Selected $m_1$` = sum(model_selected==1),
    `Selected $m_2$` = sum(model_selected==2),
    `Selected $m_3$` = sum(model_selected==3),
    `Selected $m_4$` = sum(model_selected==4)) %>%
  slice(match(order_vec, `Selection Method`)) %>%   # reorder rows
  kable(format = "latex", booktabs = T) %>%
  kable_styling(latex_options = "striped")
```

D.3 Real Data Study

The code for model assessment and selection on the real-world dataset is essentially the same as the code that is used for assessment and selection on the simulation dataset. Instead of repeating the code, we just highlight the key differences.

Note that the actual dataset is not included in the electronic version of this code, since the dataset is not publicly published.

Model Assessment

```
# Setup: load in and process real data
load("~/TreeCompetition/Documents/Thesis/Simulations/Data/df_full.Rda")
species_of_interest <- df_full %>%
  group_by(species) %>%
  summarize(n=n()) %>%
  filter(n>=400) %>%   # 10 species
  pull(species)
df <- df_full %>%
  select(x, y, species, dbh, growth, biomass_comp, ncomp) %>%
  filter(species %in% species_of_interest)

# Define parameters for simulation
set.seed(700)

# Divide data into 6 spatial "strips":
# - sample sizes: 2772 3081 3905 3674 3651 2092
strip_intervals <- list(c(-300,-100),c(-100,0),c(0,100),
  c(100,200),c(200,300),c(300,500))

# (NROW, NCOL) for each "strip", for buffered CV
# Note: strips 1 & 6 are wider, so we divide them more to preserve block size
grids <- list(c(8,4),c(8,2),c(8,2),c(8,2),c(8,2),c(8,4))
NFOLDS <- 10   # Number of folds for k-fold cv
RADIUS <- 15   # Size of buffer for moving circle bootstrap
NBOOT <- 60   # Number of moving circle bootstrap samples

assign_df <- function(interval) {
  df %>% filter(x > interval[1] & x < interval[2])
}
strip_list <- lapply(strip_intervals, assign_df)

# For each strip: use strip as training set, use other 5 as test sets
err_df <- data.frame()
for (i in 1:6) {
  df_train <- strip_list[[i]]
  NROW <- grids[[i]][1]
  NCOL <- grids[[i]][2]

  ...   # rest of code is same as model assessment simulation

}
```

The code below generates Table 5.3.

```
err_df %>%
  rename(
    `Training Error` = training_err,
    `K-fold CV` = kfold_err,
    `LOO Bootstrap` = LOO_err,
    `Buffered Grid CV` = buff_err,
    `Moving Circle Bootstrap` = moving_err) %>%
  gather(`Assessment Method`, value, `K-fold CV`:`Training Error`) %>%
  group_by(`Assessment Method`) %>%
  summarize(`Average Error Estimate` = mean(value)) %>%
  arrange(factor(`Average Error Estimate`)) %>%
  kable(
    format = "latex",
    digits = 3,
    booktabs = T)
```

The code below generates Figure C.4. Note that this is effectively the same code that is used to generate Figure 5.3.

```
err_df %>%
  gather(est_type, est_value, c(buff_err,kfold_err:training_err)) %>%
  gather(err_type, err_value, err1:err6) %>%
  mutate(
    dev = abs(est_value-err_value),
    est_type = factor(est_type, levels=c("training_err", "kfold_err",
      "LOO_err", "buff_err", "moving_err"))) %>%
  filter(substr(err_type,4,4)!=set_number) %>%
  ggplot(aes(x=err_type, y=dev, color=est_type,
    group=est_type, order=est_type)) +
  geom_line(size=1) +
  geom_point(size=3, shape=1) +
  facet_wrap(~set_number) +
  scale_color_discrete(labels=c("Training Error", "K-fold CV",
    "LOO Bootstrap", "Buffered Grid CV",  "Moving Circle \nBootstrap")) +
  scale_x_discrete(labels=1:6) +
  labs(
    y="|Error Estimate - True Error on Facet Strip|",
    x="Validation Set Strip Number",
    color="Error Estimator")
```

Model Selection

```
# Define parameters for simulation
set.seed(800)

# Divide data into 6 spatial "strips":
# - sample sizes: 2772 3081 3905 3674 3651 2092
```

```
strip_intervals <- list(c(-300,-100),c(-100,0),c(0,100),
  c(100,200),c(200,300),c(300,500))

# (NROW, NCOL) for each "strip", for buffered CV
# Note: strips 1 & 6 are wider, so we divide them more to preserve block size
grids <- list(c(8,4),c(8,2),c(8,2),c(8,2),c(8,2),c(8,4))
NFOLDS <- 10   # Number of folds for k-fold cv
RADIUS <- 15   # Size of buffer for moving circle bootstrap
NBOOT <- 60   # Number of moving circle bootstrap samples

assign_df <- function(interval) {
  df %>% filter(x > interval[1] & x < interval[2])
}
strip_list <- lapply(strip_intervals, assign_df)

# For each strip: use strip as training set, use other 5 as test sets
sel_df <- data.frame()
for (i in 1:6) {
  df_train <- strip_list[[i]]
  NROW <- grids[[i]][1]
  NCOL <- grids[[i]][2]

  model_formula <- as.formula("growth ~ species + dbh + biomass_comp + ncomp")
  model_formulas <- list(
    as.formula("growth ~ dbh + ncomp"),
    as.formula("growth ~ dbh + biomass_comp"),
    as.formula("growth ~ dbh + biomass_comp + ncomp"),
    as.formula("growth ~ species + dbh + ncomp"),
    as.formula("growth ~ species + dbh + biomass_comp"),
    as.formula("growth ~ species + dbh + biomass_comp + ncomp"))

  ...   # rest of code is same as model selection simulation

}
```

The code below generates Table 5.4.

```
order_vec <- c("sel_train", "sel_cv", "sel_LOO",
  "sel_grid_buffer", "sel_moving")
sel_df %>%
  gather(`Selection Method`, model_selected, sel_train:sel_moving) %>%
  group_by(`Selection Method`) %>%
  summarize(
    `Selected $m_1$` = sum(model_selected==1),
    `Selected $m_2$` = sum(model_selected==2),
    `Selected $m_3$` = sum(model_selected==3),
    `Selected $m_4$` = sum(model_selected==4),
    `Selected $m_5$` = sum(model_selected==5),
    `Selected $m_6$` = sum(model_selected==6)) %>%
  slice(match(order_vec, `Selection Method`)) %>%   # reorder rows
```

```
  kable(format = "latex", booktabs = T) %>%
  kable_styling(latex_options = "striped")
```

D.4 *Elements of Statistical Learning* Replication

The code in this section was used to replicate the simulation in [23], as described in Section 2.4.

Setup

```
require(leaps)   # Best subset regression
require(ModelMetrics)

# Function to generate 80x20 data frames
gen_df <- function(){
  df <- data.frame(matrix(runif(1600, min=0, max=1), ncol=20)) %>%
    mutate(Y = rowSums(.[1:10]) > 5)
  return(df)
}
# Data frame used to compute "true" error
truth <- data.frame(matrix(runif(20000, min=0, max=1), ncol=20)) %>%
  mutate(Y = rowSums(.[1:10]) > 5)

# Method to generate expected test error, given a model formula
err_true_exp <- function(model_formula) {
  errs <- vector(mode="numeric", length=100)
  for (i in 1:100) {
    df <- gen_df()
    lm <- glm(model_formula, data=df)
    errs[i] <- mse(predict(lm, truth), truth$Y)
  }
  return(mean(errs))
}
```

CV and Bootstrap Methods for Replication

Note that the methods here return MSE, not RMSE.

```
# CV method - return MSE
cv_lm <- function(model_formula, nfolds, df) {
  cv_errors <- vector(mode="numeric", length=nfolds)

  # Shuffle data and partition folds
  temp <- df[sample(nrow(df)),]
```

```
  folds <- cut(seq(1,nrow(temp)),
               breaks=nfolds,
               labels=FALSE)
  for(i in 1:nfolds){
    # Split training/testing sets
    test_index <- which(folds==i, arr.ind=TRUE)
    test <- temp[test_index, ]
    train <- temp[-test_index, ]
    # Fit lm
    fitted_model <- glm(model_formula, data=train)
    # Store error
    diffs <- predict(fitted_model, test) - test$Y
    cv_errors[i] <- mean(diffs^2)
  }
  return(mean(cv_errors))
}
```

```
# Bootstrap method - return MSE
boot_lm <- function(model_formula, df, nrep=100) {
  boot_errors <- vector(mode="numeric", length=nrep)

  for(i in 1:nrep){
    # Generate bootstrap sample
    samp <- df %>%
      sample_n(size=nrow(df), replace=TRUE)
    # Fit lm
    fitted_model <- glm(model_formula, data=samp)
    # Store error
    diffs <- predict(fitted_model, samp) - samp$Y
    boot_errors[i] <- mean(diffs^2)
  }
  return(mean(boot_errors))
}
```

```
# LOO bootstrap method - return MSE
bootLOO_lm <- function(model_formula, df, nrep=100) {
  bootLOO_errors <- vector(mode="numeric", length=nrow(df))

  for(boot in 1:nrep){
    # Generate bootstrap sample
    samp_indices <- sample(1:80, 80, replace=TRUE)
    samp <- df[samp_indices,]
    # Fit lm
    fitted_model <- glm(model_formula, data=samp)

    # Store errors on each original observation
    obs_errs <- vector(mode="numeric", length=nrow(df))
    count <- 0
    for(obs in 1:nrow(df)){
```

```
      if(!(obs %in% samp_indices)){   # if observation is not in sample
        test <- df[obs,]
        diff <- predict(fitted_model, test) - test$Y
        obs_errs[obs] <- diff^2
        count <- count + 1
      }
    }
    # Note: can include check to ensure count != 0
    bootLOO_errors[boot] <- sum(obs_errs)/count
  }
  return(mean(bootLOO_errors))
}
```

```
# .632 bootstrap method - return MSE
boot632_lm <- function(model_formula, df, nrep=100){
  bootLOO_errors <- vector(mode="numeric", length=nrow(df))
  training_errors <- vector(mode="numeric", length=nrow(df))

  for(boot in 1:nrep){
    # Generate bootstrap sample
    samp_indices <- sample(1:80, 80, replace=TRUE)
    samp <- df[samp_indices,]
    # Fit lm
    fitted_model <- glm(model_formula, data=samp)

    # Store errors on each original observation
    obs_errs <- vector(mode="numeric", length=nrow(df))
    count <- 0
    for(obs in 1:nrow(df)){
      if(!(obs %in% samp_indices)){   # if observation is not in sample
        test <- df[obs,]
        diff <- predict(fitted_model, test) - test$Y
        obs_errs[obs] <- diff^2
        count <- count + 1
      }
    }
    # Note: can include check to ensure count != 0
    bootLOO_errors[boot] <- sum(obs_errs)/count
    training_errors[boot] <- mse(predict(fitted_model, samp), samp$Y)
  }
  return(.632*mean(bootLOO_errors) + .368*mean(training_errors))
}
```

Run Simulations

`compute_errs` outputs a data frame with 8 columns and 20 rows. The columns contain the values of p, 10-fold CV error, LOOCV error, bootstrap error, LOO bootstrap error,

.632 bootstrap error, true expected error, and true conditional error. The rows contain the results for $p = 1, 2, \ldots, 20$.

```
compute_errs <- function(){
  # Generate random data frame
  df <- gen_df()
  output_matrix <- matrix(0L, nrow=20, ncol=8)

  for (p in 1:20) {
    # Get model formula for best lm with p predictors
    model <- regsubsets(x=df[,-21], y=df[,21], nvmax=p)
    vars <- summary(model)$which[p,]    # p predictors
    bestvars <- names(vars)[vars]
    model_formula <- as.formula(paste("Y ~",
      paste(bestvars[-1], collapse="+")))
    # Fit best lm (used for true conditional error)
    lm <- glm(model_formula, data=df)

    # Generate error estimates
    err_cv10 <- cv_lm(model_formula, 10, df)
    err_cvLOO <- cv_lm(model_formula, 80, df)
    err_boot <- boot_lm(model_formula, df)
    err_bootLOO <- bootLOO_lm(model_formula, df)
    err_boot632 <- boot632_lm(model_formula, df)
    err_true_exp <- err_true_exp(model_formula)
    err_true_cond <- mse(predict(lm, truth), truth$Y)

    output_matrix[p,] <- c(
      p, err_cv10, err_cvLOO,
      err_boot, err_bootLOO, err_boot632,
      err_true_exp, err_true_cond)
  }
  output_df <- data.frame(output_matrix)
  colnames(output_df) <- c(
      "p", "err_cv10", "err_cvLOO",
      "err_boot", "err_bootLOO", "err_boot632",
      "err_true_exp", "err_true_cond")
  return(output_df)
}
```

This loop produces results for use in both simulations.

```
df_errs <- data.frame(matrix(ncol = 8, nrow = 0))
for (i in 1:100){
  df_errs <- rbind(df_errs, compute_errs())
}
colnames(df_errs) <- c("p", "CV10", "CVLOO", "Boot", "BootLOO", "Boot632",
  "TrueExp", "TrueCond")
```

Analyzing Results

The code below generates Figure 2.4.

```
df_errs %>%
  select(-CVLOO) %>%
  gather(method, error, CV10:Boot632) %>%
  group_by(p, method, TrueExp, TrueCond) %>%
  summarize(error = mean(error)) %>%
  ungroup() %>%
  mutate(
    ErrCond = abs(error-TrueCond),
    ErrExp = abs(error-TrueExp)) %>%
  group_by(p, method) %>%
  summarize(
    ErrCond = mean(ErrCond),
    ErrExp = mean(ErrExp)) %>%
  mutate(method=factor(method,
    labels=c("Bootstrap", ".632 Bootstrap", "LOO Bootstrap",
             "10-fold CV"))) %>%
  gather(ErrType, Value, ErrCond:ErrExp) %>%
  mutate(ErrType=factor(ErrType,
    labels=c("Conditional", "Expected"))) %>%
  ggplot(aes(x=p, group=interaction(method, ErrType), color=method)) +
  geom_line(aes(y=Value, lty=ErrType), size=1) +
  labs(
    x="p (# predictors in best subset regression)",
    y="|Estimate - True Error|",
    color="Error Estimation \nMethod",
    lty="True Error Type")
```

The code below generates Figure 2.5.

```
df_errs %>%
  group_by(p) %>%
  summarize(
    CV10 = cor(CV10, TrueCond),
    CVLOO = cor(CVLOO, TrueCond),
    Boot = cor(Boot, TrueCond),
    BootLOO = cor(BootLOO, TrueCond),
    Boot632 = cor(Boot632, TrueCond)
  ) %>%
  gather(method, cor, CV10:Boot632) %>%
  mutate(method=factor(method,
    labels=c("Bootstrap", ".632 Bootstrap", "LOO Bootstrap",
             "10-fold CV", "LOOCV"))) %>%
  ggplot(aes(x=p, y=cor, group=method, color=method)) +
  geom_line() +
  geom_abline(slope=0, intercept=0) +
  labs(
```

```
    x="p (# predictors in best subset regression)",
    y="Correlation",
    color="Error Estimation \nMethod")
```

D.5 Spatial Bootstrap Simulation

We only show the code for generating Figures A.5 and A.6, since the code for generating the non-toroidally wrapped bootstraps is effectively the same.

Setup

```
# Define parameters: uniform distribution over height x width region
n_obs <- 1000
width <- 100
height <- 100
x <- runif(n_obs, 0, width)
y <- runif(n_obs, 0, height)
index <- c(1:n_obs)
df <- data.frame(x,y,index)

# Randomly sample tile of width `size` from `df`, w/ toroidal wrapping
# Returns indices of sampled points
sample_tile_tw <- function(df, size=10) {
  x_coord <- runif(1, 0, width)
  y_coord <- runif(1, 0, height)
  temp <- df %>%
    filter(
      x > x_coord-(size/2) | x < x_coord-width+(size/2),
      y > y_coord-(size/2) | y < y_coord-height+(size/2),
      x < x_coord+(size/2) | x > x_coord+width-(size/2),
      y < y_coord+(size/2) | y > y_coord+height-(size/2)
    )
  return(temp$index)
}

# Runs one moving tile bootstrap with `nboot` iterations
# Returns indices of sampled points
tile_bootstrap <- function(df, nboot, size=10, tw=FALSE) {
  sample <- c()
  if (tw) {
    for (i in 1:nboot) {
      sample <- c(sample, sample_tile_tw(df, size))
    }
  }
  else {
    for (i in 1:nboot) {
```

```
      sample <- c(sample, sample_tile(df, size))
    }
  }
  return(sample)
}
```

Run Simulation

For each of the `numsim` iterations, we store a matrix of the proportion of unique points (out of all points in the original sample) that are contained in a bootstrap sample, gridding over the block sizes in `block_props` and the numbers of bootstrap iterations in `nums_boot`. This code takes about four hours to run.

```
numsim <- 100
block_props <- c(1:5)
nums_boot <- c(5,10,15,20,25,30,40,50,75,100)
matrix_list <- list()
for (i in 1:numsim) {
  spatial_props <- matrix(0L, nrow=10, ncol=5)
  for (block_prop in block_props) {
    block_size <- block_prop*0.1*width
    props <- vector(mode="numeric", length=5)
    count <- 1
    for (num_boot in nums_boot) {
      props[count] <- length(unique(
        tile_bootstrap(df, num_boot, size=block_size, tw=TRUE)))
      count <- count+1
    }
    spatial_props[,block_prop] <- props
  }
  matrix_list[[i]] <- spatial_props
}
```

```
# Helper function to turn matrix_list into tidy df
process_matrix <- function(matrix) {
  df <- data.frame(matrix)
  colnames(df) <- c(0.1,0.2,0.3,0.4,0.5)
  df <- df %>%
    gather(block_size, samp_size, `0.1`:`0.5`) %>%
    mutate(block_size = as.numeric(block_size)) %>%
    bind_cols(data.frame(
      rep(c(5,10,15,20,25,30,40,50,75,100), 5)))
  df <- df[,c(1,3,2)]
  colnames(df) <- c("block_size", "num_samp", "samp_size")
  return(df)
}
```

```
df_list <- lapply(matrix_list, process_matrix)

# Combine all data frames into one data frame
require(data.table)
output <- rbindlist(df_list)
```

Analyzing Results

The code below generates to generate Figure A.5.

```
# Color scale for discrete gradient from blue to orange
cc <- scales::seq_gradient_pal("blue", "orange", "Lab")(seq(0,1,length.out=5))

output %>%
  mutate(
    samp_size = samp_size/1000,
    block_size=as.factor(block_size)) %>%
  group_by(block_size, num_samp) %>%
  summarize(
    samp_size = mean(samp_size)) %>%
  ggplot(aes(x=num_samp, y=samp_size, group=block_size, color=block_size)) +
  geom_line() +
  scale_color_manual(values=cc) +
  labs(
    x="Number of Bootstrap Samples",
    y="Prop. of Points in Sample",
    color="Block Width \n(as prop. of \ntotal width)")
```

The code below generates Figure A.6.

```
output %>%
  mutate(
    samp_size = samp_size/1000,
    block_size=as.factor(block_size)) %>%
  group_by(block_size, num_samp) %>%
  summarize(
    sd = sd(samp_size),
    samp_size = mean(samp_size)) %>%
  ggplot(aes(x=num_samp, y=sd, group=block_size, color=block_size)) +
  geom_line() +
  scale_color_manual(values=cc) +
  labs(
    x="Number of Bootstrap Iterations",
    y="SD of Prop. of Points in Sample",
    color="Block Width \n(as prop. of \ntotal width)")
```

The code below generates Figure A.4(a). The code for Figure A.4(b) is identical, with `tw` set to T.

```
set.seed(3)
indices <- tile_bootstrap(df, nboot=15, size=30, tw=F)
ggplot(df, aes(x=x, y=y)) +
  geom_point(aes(color=(index %in% indices))) +
  labs(
    x="X",
    y="Y",
    color="Point in \nBoostrap \nSample?"
  )
```

D.6 "Flipped" CV Simulation

The code for the "flipped" CV simulations is essentially identical to code used in the model assessment and selection simulations. For "flipped" CV assessment, we use the random seed 200 for the $N = 500$ case, and 700 for the $N = 5000$ case. For "flipped" CV selection, we use the random seed 800 for the $N = 500$ case, and 300 for the $N = 5000$ case.

Bibliography

[1] Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

[2] Miguel B Araújo, Richard G Pearson, Wilfried Thuiller, and Markus Erhard. Validation of species–climate impact models under climate change. *Global Change Biology*, 11(9):1504–1513, 2005.

[3] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

[4] Volker Bahn and Brian J McGill. Testing the predictive performance of distribution models. *Oikos*, 122(3):321–331, 2013.

[5] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of K-fold cross-validation. *Journal of Machine Learning Research*, 5(Sep):1089–1105, 2004.

[6] Simone Borra and Agostino Di Ciaccio. Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational Statistics & Data Analysis*, 54(12):2976–2989, 2010.

[7] Leo Breiman and Philip Spector. Submodel selection and evaluation in regression. the x-random case. *International Statistical Review*, pages 291–319, 1992.

[8] Alexander Brenning. Spatial cross-validation and bootstrap for the assessment of prediction rules in remote sensing: The R package sperrorest. In *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pages 5372–5375. IEEE, 2012.

[9] Prabir Burman, Edmond Chow, and Deborah Nolan. A cross-validatory method for dependent data. *Biometrika*, 81(2):351–358, 1994.

[10] Edward Carlstein. The use of subseries values for estimating the variance of a general statistic from a stationary sequence. *The Annals of Statistics*, pages 1171–1179, 1986.

[11] Patrick S Carmack, William R Schucany, Jeffrey S Spence, Richard F Gunst, Qihua Lin, and Robert W Haley. Far casting cross-validation. *Journal of Computational and Graphical Statistics*, 18(4):879–893, 2009.

[12] C-K Chu and James Stephen Marron. Comparison of two bandwidth selectors with dependent errors. *The Annals of Statistics*, pages 1906–1918, 1991.

[13] Chang-Jo Chung and Andrea G Fabbri. Predicting landslides for risk analysis - spatial models tested by a cross-validation technique. *Geomorphology*, 94(3-4):438–452, 2008.

[14] Chang-Jo F Chung and Andrea G Fabbri. Validation of spatial prediction models for landslide hazard mapping. *Natural Hazards*, 30(3):451–472, 2003.

[15] Christopher Daly. Guidelines for assessing the suitability of spatial climate data sets. *International Journal of Climatology*, 26(6):707–721, 2006.

[16] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap Methods and their Application*, volume 1. Cambridge university press, 1997.

[17] B Efron et al. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[18] Bradley Efron. Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[19] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference*, volume 5. Cambridge University Press, 2016.

[20] Bradley Efron and Robert Tibshirani. Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.

[21] Roberts et al. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 2017.

[22] Carsten F Dormann, Jana M McPherson, Miguel B Araújo, Roger Bivand, Janine Bolliger, Gudrun Carl, Richard G Davies, Alexandre Hirzel, Walter Jetz, W Daniel Kissling, et al. Methods to account for spatial autocorrelation in the analysis of species distributional data: a review. *Ecography*, 30(5):609–628, 2007.

[23] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics New York, 2001.

[24] Seymour Geisser. A predictive approach to the random effect model. *Biometrika*, 61(1):101–107, 1974.

[25] JN Goetz, Alexander Brenning, H Petschko, and P Leopold. Evaluating machine learning and statistical prediction techniques for landslide susceptibility modeling. *Computers & Geosciences*, 81:1–11, 2015.

[26] Peter Hall. Resampling a coverage pattern. *Stochastic Processes and their Applications*, 20(2):231–246, 1985.

[27] Wolfgang Härdle, Peter Hall, and James Stephen Marron. How far are automatically chosen regression smoothing parameters from their optimum? *Journal of the American Statistical Association*, 83(401):86–95, 1988.

[28] Jeffrey D Hart and Seongbaek Yi. One-sided cross-validation. *Journal of the American Statistical Association*, 93(442):620–631, 1998.

[29] Robert J Hijmans. Cross-validation of species distribution models: removing spatial sorting bias and calibration with a null model. *Ecology*, 93(3):679–688, 2012.

[30] Jin Huang and Charles X Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, 2005.

[31] Jerald B Johnson and Kristian S Omland. Model selection in ecology and evolution. *Trends in Ecology & Evolution*, 19(2):101–108, 2004.

[32] Yoonsuh Jung. Multiple predicting K-fold cross-validation for model selection. *Journal of Nonparametric Statistics*, pages 1–19, 2017.

[33] Ji-Hyun Kim. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, 53(11):3735–3745, 2009.

[34] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[35] Hans R Kunsch. The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, pages 1217–1241, 1989.

[36] Selmer C Larson. The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology*, 22(1):45, 1931.

[37] Kévin Le Rest, David Pinaud, Pascal Monestiez, Joël Chadoeuf, and Vincent Bretagnolle. Spatial leave-one-out cross-validation for variable selection in the presence of spatial autocorrelation. *Global Ecology and Biogeography*, 23(7):811–820, 2014.

[38] Pierre Legendre. Spatial autocorrelation: trouble or new paradigm? *Ecology*, 74(6):1659–1673, 1993.

[39] Jack J Lennon. Red-shifts and red herrings in geographical ecology. *Ecography*, 23(1):101–113, 2000.

[40] Regina Y Liu and Kesar Singh. Moving blocks jackknife and bootstrap capture weak dependence. *Exploring the Limits of Bootstrap*, 225:248, 1992.

[41] Colin L Mallows. Some comments on C_p. *Technometrics*, 15(4):661–675, 1973.

[42] James Stephen Marron. Partitioned cross-validation. *Econometric Reviews*, 6(2):271–283, 1987.

[43] Annette M Molinaro, Richard Simon, and Ruth M Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 2005.

[44] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. In *Advances in Neural Information Processing Systems*, pages 307–313, 2000.

[45] Jonne Pohjankukka, Tapio Pahikkala, Paavo Nevalainen, and Jukka Heikkonen. Estimating the prediction performance of spatial models via spatial K-fold cross validation. *International Journal of Geographical Information Science*, 31(10):2001–2019, 2017.

[46] Dimitris N Politis and Joseph P Romano. A circular block-resampling procedure for stationary data. *Exploring the Limits of Bootstrap*, pages 263–270, 1992.

[47] Dimitris N Politis and Joseph P Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994.

[48] Jeff Racine. Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of Econometrics*, 99(1):39–61, 2000.

[49] Juan D Rodriguez, Aritz Perez, and Jose A Lozano. Sensitivity analysis of K-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):569–575, 2010.

[50] Gideon Schwarz et al. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[51] PAGE Segurado, Miguel B Araujo, and WE Kunin. Consequences of spatial autocorrelation for niche-based models. *Journal of Applied Ecology*, 43(3):433–444, 2006.

[52] Jun Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88(422):486–494, 1993.

[53] Jun Shao. Bootstrap model selection. *Journal of the American Statistical Association*, 91(434):655–665, 1996.

[54] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974.

[55] RJ Telford and HJB Birks. The secret assumption of transfer functions: problems with spatial autocorrelation in evaluating model performance. *Quaternary Science Reviews*, 24(20-21):2173–2179, 2005.

[56] RJ Telford and HJB Birks. Evaluation of transfer functions in spatially structured environments. *Quaternary Science Reviews*, 28(13-14):1309–1316, 2009.

[57] Waldo R Tobler. A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46(1):234–240, 1970.

[58] Gitte Vanwinckelen and Hendrik Blockeel. On estimating model accuracy with repeated cross-validation. In *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, pages 39–44, 2012.

[59] Samuel D Veloz. Spatially autocorrelated sampling falsely inflates measures of accuracy for presence-only niche models. *Journal of Biogeography*, 36(12):2290–2299, 2009.

[60] Sonja Wehberg and Martin Schumacher. A comparison of nonparametric error rate estimation methods in classification problems. *Biometrical Journal*, 46(1):35–47, 2004.

[61] Seth J Wenger and Julian D Olden. Assessing transferability of ecological models: an underappreciated aspect of statistical validation. *Methods in Ecology and Evolution*, 3(2):260–267, 2012.

[62] Tzu-Tsung Wong. Performance evaluation of classification algorithms by K-fold and leave-one-out cross validation. *Pattern Recognition*, 48(9):2839–2846, 2015.

[63] Yuhong Yang. Comparing learning methods for classification. *Statistica Sinica*, pages 635–657, 2006.

[64] Yuhong Yang et al. Consistency of cross validation for comparing regression procedures. *The Annals of Statistics*, 35(6):2450–2473, 2007.

[65] Ping Zhang. Model selection via multifold cross validation. *The Annals of Statistics*, pages 299–313, 1993.

[66] Yongli Zhang and Yuhong Yang. Cross-validation for selecting a model selection procedure. *Journal of Econometrics*, 187(1):95–112, 2015.

Corrections

When originally submitted, this honors thesis contained some errors which have been corrected in the current version. Here is a list of the errors that were corrected.

Various places in the thesis. Approximately 5 typographic errors were corrected in the thesis proper, and 40 in the bibliography. Commas were added to "i.e." in 19 locations and "e.g." in 12 locations.

Other changes:

Title Page. Added "and Statistics" to department name.

p. 2, l. 6. Removed titles from "Albert Y. Kim" and "David Allen".

p. 3. Added footnote 1.

p. 5. Added "(see page 168 in [23])" to footnote 5.

p. 11, l. 9 Removed "our assessment methods" from after "we want" in first sentence on line. Removed "On one hand," from beginning of second sentence on line.

p. 11, l. 11. Replaced "On the other hand," with "Yet."

p. 11, l. 14. Added reference to [23].

p. 11. Rewrote final full paragraph on Page 11 to include more description of the simulation procedure. Added "We simulate... for each model and dataset." to the middle of the paragraph and removed "Using best subset linear regression models, the simulation compares the error estimates from CV and the bootstrap to the models' true conditional and expected errors." from the beginning of the paragraph.

p. 11, l. –1. Added paragraph break.

p. 12, l. 1. Added "that the dotted lines generally lie below their corresponding solid lines, i.e.," to middle of sentence.

p. 13, l. 5. Added reference to [23].

p. 14, l. –5. Added "we will see that" after "In general."

p. 25, l. –6. Replaced "insignificant" with "non-significant."

p. 26, l. 1. Replace "will focus" with "focuses."

p. 27, l. 4. Added reference to [37].

p. 27, l. 6. Added reference to [21].

p. 27, l. –4. Added reference to [Kim and Allen, in progress]. Moved "See Figure 3.5 for a visual" to footnote.

p. 28, l. 4. Added "in R" after "`sperrorest` package."

p. 29, l. –7. Replaced "provides an early study of the theoretical properties of" with "introduces and studies."

p. 30, l. 7. Removed "The theory behind" from beginning of sentence.

p. 31. In footnote 7, replaced "I have found no such suggestions in the literature, and I will not explore it" with "the literature on this idea is sparse and will not be explored."

p. 33, l. –1. Added reference to [3].

p. 34, l. –5. Added reference to [3].

p. 37, l. 15–16. Converted sentence from first-person to passive voice.

p. 37, l. 16. Removed "though it seems doubtful that it would not exist" from end of paragraph.

p. 37, l. 17. Converted sentence from first-person to passive voice.

p. 38, l. −9. Replaced "I have been unable to locate much" with "there is sparsity in the."

p. 38, l. −3. Removed sentence: "Of course, further research may uncover significant literature on CV and the bootstrap for spatial model selection, but for now, I can only acknowledge that I have not found it."

p. 38, l. −2. Replaced "room for studying" with "opportunity to study."

p. 39. Replaced "choose" with "chose" in footnote 2.

p. 42, l. −6. Removed titles from "Albert Y. Kim" and "David Allen."

p. 43. Increased size of Figure 5.3.

p. 44, l. 3. Replaced sentence "While Table 5.3 shows spatial methods outperforming nonspatial methods on average, they do not consistently do so for any particular strip" with "Table 5.3 shows how spatial methods are more pessimistic than nonspatial methods, which generally leads to estimates closer to true error. In cases such as strip 3, however, the spatial methods can be too pessimistic, which leads the nonspatial methods to be closer to true error."

p. 44, l. 7. Replaced "is much room" with "are significant opportunities."

p. 44, l. 8. Removed "though such study lies beyond the scope of this thesis" from end of sentence.

p. 45, l. −1. Replaced sentence "To account for this fact, we suggest consulting learning curves as one changes different simulation parameters, to make sure that models continue to require significant amounts of training data before minimizing their validation errors" with "To account for this fact, we suggest consulting learning curves as one changes different simulation parameters. The models used in the simulations should always require significant amounts of training data before minimizing their validation errors."

p. 47, l. 12 Removed "using results from the literature on time-series resampling to build our intuitions for how spatial resampling methods may behave" from end of sentence.