Trustworthy AI During Machine Learning Model Development: Enhancing Explainability and Robustness of ML Models with Fitting Graphs

Robbie T. Nakatsu

Department of Information Systems and Business Analytics,
Loyola Marymount University, Los Angeles, CA, 90066, USA

Email: Robbie.Nakatsu@lmu.edu
ORCID: 0000-0003-1523-4224

Abstract. This study demonstrates how a fitting graph can enhance explainability and robustness during the model development phase of a machine learning project. The approach is illustrated with a ridge regression task, where the goal is to identify the best-fitting regularization parameter, λ, from a range of values. A simple scatterplot of λ values (indicating model complexity) against average mean squared error, MSE (representing predictive accuracy), provides a visual representation to help the model developer determine if sufficient iterations of k-fold cross-validation have been performed. In addition, this study shows how fitting graph curves can be estimated and constructed from noisy scatterplots using regression splines. Instead of increasing the number of reps of cross-validation, a regression spline can save you time in estimating the fitting graph, using far fewer iterations.
The fitting graph is also presented as a tool to promote model robustness, defined as the model's ability to maintain performance levels across variations in the hyperparameter λ. This concept is demonstrated through a case study on an unstable polynomial regression model. The simulation study reveals that standard k-fold cross-validation, even when repeated 5 or 10 times, selects an incorrect and unstable λ by an overwhelming margin. In contrast, the fitting graph method reliably selects a λ that is both well-fitting and stable. Without the fitting graph, the model developer is led astray and is more likely to choose a highly unstable λ, leading to suboptimal model performance.

Keywords: machine learning, explainable AI, model robustness, model validation, hyperparameter tuning, ridge regression modeling, regularization, fitting graphs

Author's Statement:
The author has no competing interests to declare that are relevant to the content of this article.
The author performed all the analysis of the data and wrote the paper as a solo author.
The datasets analysed during the current study are available from the author by request. The author will also post all datasets and programs on a website before the publication date.

Trustworthy AI During Machine Learning Model Development: Enhancing Explainability and Robustness of ML Models with Fitting Graphs

1 Introduction

The need for trustworthy AI has become a critical research topic due to AI's growing influence on society. While AI systems, especially machine learning models, can provide fast and accurate predictions, they sometimes yield incorrect results. For instance, generative AI tools like ChatGPT have been known to produce plausible but false information, or "hallucinations," making it hard for users to distinguish accurate information from misinformation. A machine learning model used in pretrial detention decisions was found to be biased against Black defendants [40]. Epic Systems' AI model for predicting sepsis, the leading cause of death in hospitals, failed to detect 67% of sepsis cases and falsely flagged 88% of non-sepsis patients [41]. These examples show that the trustworthiness of many AI systems remain a central challenge.

What are the components of Trustworthy AI? Trustworthy AI is a broad and multi-dimensional concept, encompassing consideration of several issues. In recent years, a number of frameworks have been proposed to help designers and developers build trustworthy AI systems(e.g., [11,21,25,32,40]). Although there are some differences as to what constitutes trustworthy AI, and what the most important aspects are, these frameworks commonly consider the following five dimensions:

- **Performance:** AI demonstrates competence in performing its task, such as the predictive accuracy of the machine learning model**.**
- **Explainability**: AI is transparent and open to inspection. This means that stakeholders can probe the AI system to understand how it makes its decisions.
- **Robustness**: AI is reliable and resilient, performing accurately over a wide range of conditions and scenarios.[1]
- **Fairness**: AI is free from bias and discrimination.
- **Ethical Considerations**: AI is aligned to society's goals, including respect for human rights and values.

It goes without saying that AI systems should exhibit strong performance, otherwise they cannot be trusted. Hence, it is important for AI models to be evaluated on appropriate performance benchmarks such as predictive accuracy, F1 score, AUC, precision, and recall. After all, a predictive machine learning model is only as good as how well it can make its predictions, free from errors. This research centers on two related but distinct dimensions to performance, namely, explainability and robustness.

Explainability and robustness are tied to performance, because they address different aspects on performance: on the one hand, the ability to explain performance (explainability), or why a model performed in the way that it did, and on other the hand, the ability to sustain performance levels across a wide range of conditions (robustness). Fairness and ethical considerations, though important dimensions, are not addressed in this study.

Other components of trustworthy AI, not mentioned above, include **accountability** [21,25,32], **safety** [21,32], **sustainability** [21,32], **privacy** [21,32], **security** [21,25,32], **lawfulness** [11], and **reproducibility**

[1] Robustness is sometimes referred to as reliability.

[25], among others. Sometimes the definitions of these concepts are intertwined with each other and not treated as separate and distinct. For example, safety is frequently mentioned in conjunction with robustness because having safe AI presumes that the AI system can perform well under varying operational conditions. Accountability involves justifying the decisions the AI system is making, so it is frequently associated with explainability—after all, justifying decisions means explaining how decisions are made. Fairness, privacy, sustainability, and security are sometimes embedded in broader discussions of ethical AI.

The first dimension we focus on is **explainability**. A large body of research on Explainable AI (XAI), also known as interpretable AI, explores how AI systems can explain their actions and decisions in ways humans can understand [6]. A common complaint leveled against AI systems is that they are black boxes that are hard to understand. When an AI's decisions do not make sense to the end-user, it is hard to understand what factors determined its decision. Explainable AI attempts to make the decision-making process more transparent so that an end-user can use the system's outputs in more thoughtful and critical ways. Explanations in AI is not a new research topic but has been a decades-old concern. It was notably a popular topic during the peak years of Expert Systems development, in the 1970s and 1980s, when researchers were looking for ways to provide explanations for rule-based expert systems [5,7,37]. But the need for Explainable AI has emerged once again as an urgent topic given the problems and limitations associated with AI usage.

Moreover, explainability more broadly considers how to make the entire lifecycle of the AI systems development more transparent, in terms of the steps on how the AI system was created, tested, and implemented. To foster system transparency, a variety of information could be disclosed including "the design purposes, data sources, hardware requirements, configuations, working conditions, expected usage, and system performance" [25]. Hence, explainability entails the generation of documentation in all phases of AI systems development.

Most previous research on Explainable AI has primarily focused on the end-user, aiming to help them interpret the system's outputs. Hence, Explainable AI has dealt with human-centered design, and how to deliver explanations in a way that is easy to process and understand. Much of the past research has focused on *post-hoc* explanations, or explanations provided to the end-user after the machine learning model is deployed. For example, an important research area in explainable AI has considered **feature importance**, or how the features of a machine learning model contribute to the final prediction so that a non-expert end-user can better understand the rationale underlying an AI system's decision. Subianto and Siebes [36] describe an approach that provides insights on the importance of a features. Each feature in the AI model is assigned a weight that refects the feature's overall influence on the prediction. Cortes and Embrechts [8] describe a visualization approach based on a sensitivity analysis method. Their approach measures the effects on the outcome when the input features are varied along a range of values. Goldstein et al. [15] considers individual conditional expectation (ICE) plots. These plots display the average prediction of an AI model when an individual feature varies over its range. This research stream considers the end-user who is trying to understand an AI's model outputs and how it makes its decisions.

By contrast, this study focuses on a different stakeholder, the **model developer**, who is tasked to design, evaluate, and select the correct machine learning model. Indeed, as Dhanorkar et al. [10] notes, "An AI system does not exist in a social vacuum. An AI system deployed in the real world has a wide range of stakeholders that extends beyond the immediate users (e.g., regulators, model developers, decision-makers, consumers)" (p. 1593). Each of these stakeholders may have different informational needs, based on the tasks they are required to work on. Whereas the end-user would be primarily concerned with what factors (or features) led to a system's prediction, the model developer is more concerned with understanding how to improve the AI model from a technical standpoint—e.g., how to improve its predictive accuracy and robustness. In this study, we focus on an information visualization known as the fitting graph[2], which is an explanations tool to support the model developer.

The second dimension that we focus on is **robustness**, which refers to the development of AI systems that are reliable and resilient. Braiek and Khomf [4] define robustness as the ability of the AI system to "maintain stable and reliable performance across a broad spectrum of conditions, variations, or challenges, demonstrating resilience and adaptability in the face of uncertainties or unexpected changes" (pp. 1-2). One notable example, which illustrates the importance of robust AI, are self-driving cars that need to process real-time data from sensors like cameras, radar, and lidar to navigate roads and handle unpredictable driving conditions. If these systems are to perform at a high level, they must be able to adapt to changes in road conditions, weather, and unexpected human behaviors from other vehicles.

The importance of robust AI has garnered increasing attention in recent years, given the application of AI to safety-critical areas such as self-driving vehicles, medical diagnosis, aviation guidance, among others. In response to this need, there is a large and growing body of literature that focuses on developing robust AI. Hendrycks and Dietterich [18] investigate AI systems that can handle distributional shifts, adversarial inputs, and maintain reliable performance in uncertain environments. They discuss methods to achieve robust AI. Goodfellow, Bengio, and Courville [16] address robustness in the context of developing deep learning models. They discuss how deep learning models can be made more robust to adversarial inputs, noise, and shifts in data distribution. Song et al. [34] describe robust AI as systems that maintain high performance even when its inputs are perturbed, or when they face adversarial attacks.

In addition to limiting this study to explainability and robustness, we also focus solely on the **model development** phase within the AI development lifecycle. To frame this study and provide it with some context, let's consider what phases a typical AI project moves through. Decker et al. [9] provide a typical framework that considers six phases. We condensed their modeling and evaluation phases into a single phase—model development—because modeling and evaluation frequently occur together, leaving us with five phases. Moreover, addressing explainability and robustness involves both modeling and evaluation activities. Based on their framework, and the combining of modeling and evaluation into a single phase, here are the five phases of a typical machine learning development project:

1. **Business and Data Understanding**. Business requirements are gathered and scoped, and data collection occurs.
2. **Data Preparation**. Data cleaning and preprocessing occurs. Among the tasks performed are data standardization, handling missing values, outlier detection and removal, and feature engineering.
3. **Model Development**. This is the model learning phase. It involves model selection, model training, and hyperparameter tuning of the AI model. In addition, the AI model is evaluated using appropriate performance metrics. Other issues such as explainability and robustness are considered during this phase.

[2] The end-user, in fact, may not be interested in the fitting graph, which shows how model performance varies as a function of model complexity.

4. **Deployment**. The AI system is implemented into the production environment of the organization.
5. **Monitoring and Maintenance**. This post-implementation phase considers possible changes in the production environment and monitors the performance of the AI model to ensure that it meets expected performance metrics

Li et al. [25] argue that AI trustworthiness should be established and promoted throughout the lifecycle of an AI development project. To illustrate this point, the first phase (Business and Data Understanding) involves data collection. In order to promote robustness, it is recommended to collect data that represent the diversity of situations that might arise in the real world. At the other end of the lifecycle (Monitoring and Maintenance), monitoring the machine learning model is necessary to ensure that it continues to perform well and is able to correctly make predictions on new and unseen examples that may arise. Indeed, creating trustworthy AI is an ongoing process that takes place across the lifecycle, and continues even after the AI system has been implemented and deployed in the organization.

In this study, we focus on the model developer, who may not understand whether a machine learning model is the correct one or not. The questions that we address are: How do we support the model developer with explanations? To make our ideas more concrete, this study illustrates model explainability and robustness with a machine learning task involving the creation of a linear regression model. The task involves tuning the regularization parameter λ in a linear regression model. The model developer may require some explanatory support to help guide in the selection of the correct model. How is the model developer supported in this task, so that he/she is on the right track? What type of information graphic would help a model developer trust that the correct model has been selected? How can we be sure that the linear regression model selected is a robust one, or one that is insensitive to the miscalculation of its regularization parameter?

The remainder of this paper is organized as follows: Section 2 describes the task to be solved, namely, how to find the best-fitting regularization parameter λ, such that it does not overfit or underfit a linear regression model. Section 3 describes the visualization tool, the fitting graph, which is used to provide explanations during the model development phase. Section 4 shows how to derive and actively construct fitting graphs directly from your data. First, scatterplots of model complexity plotted against model error are generated; then, fitting graphs are generated from these scatterplots by using regression splines. Section 5 describes a case study in which a highly unstable ridge regression model is analyzed. The results of a simulation study show how traditional validation methods utterly fail to find a good solution, while using a fitting graph is a more robust way of finding a well-fitting model. Section 6 provides summary discussion and conclusions of these results.

2 The Task: Finding the Best-Fitting Regularization Parameter

Linear regression is a popular technique for building a model that predicts, or estimates, a quantitative outcome. Numerous textbooks have been written on the subject. The technique involves fitting a linear model that minimizes the mean squared error (MSE) and then uses the linear model to make predictions on unseen data. Assume a linear regression model of the form

$$\mathrm{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon \quad (1)$$

where Y is the target or outcome variable, and X_1, X_2,…, X_p are the independent, or feature variables. Ordinary least squares (OLS) regression will find the β coefficients that minimize the mean squared error (MSE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 \quad (2)$$

where y_i is the actual outcome and $\widehat{y_i}$ is the predicted outcome fitted by using the regression model for i = 1 to n observations.

However, linear regression is susceptible to model overfitting. Model overfitting occurs when a regression model tries to fit the **training data** as well as possible, at the expense of poor generalization to unseen data. A graphical visualization of model overfit, involving a linear regression model, is provided by Fig. 1. In this example, a set of x values (n=50) is randomly generated from a normal distribution (μ=50, σ=10). The y values are calculated from the quadratic function: $\mathbf{y = x^2 - 5x + \epsilon}$, with the error term ϵ also randomly generated from a normal distribution (μ=0, σ=10). The data is then fit with four different regression models: (a) a straight line (underfit model), (b) a quadratic curve (correct fit), (c) a polynomial of degree 10 (overfit model), and (d) a polynomial of degree 15 (overfit model). Fig. 1 shows how models (c) and (d) overfit the data. Both models chase after noise in the data, and result in more erratic curves. When a polynomial of degree 15 is fit to the data, the curve becomes extremely erratic. Even though R^2 continues to improve with higher order polynomials, the overfit models would not generalize well to unseen data.

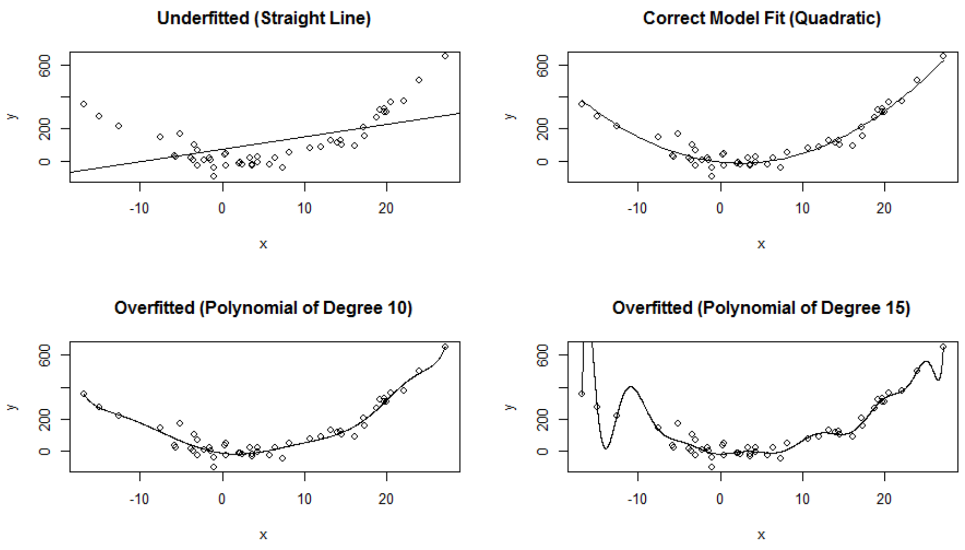
One popular technique that can reduce model overfitting is known as **ridge regression** (Hoerl and Kennard, 1970; Marquardt, 1970). The technique involves fitting a model of all predictors, like in OLS regression, but the estimated β coefficients are shrunken towards zero. Mathematically, ridge regression penalizes the β parameter estimates by adding a penalty term to the MSE in Eq. (2):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \quad (3)$$

where i is the observation from 1 to n and β_j is the coefficient of the predictor variable from j = 1 to p.

From Eq. (3), ridge regression would shrink the β estimates towards 0 as λ becomes large, because of the penalty term: To minimize MSE, you would need smaller β estimates. On the other hand, the β estimates would remain unchanged as λ approaches 0: You can see that Eq. (3) reduces to Eq. (2) when λ = 0. The effect of shrinking the coefficients (a process known as regularization) is that the regression model is less prone to model overfitting. The goal of the ridge regression modeler is to find the correct value for the λ parameter. Choosing too small a λ will result in an overfit model, while choosing too large a λ will result in an underfit model. Hence, the designer should find a λ that minimizes MSE.

Fig. 1. A visualization of model overfit on four linear regression models, with different levels of model complexity. On the upper left panel, a line underfits the data (R^2=0.269). On the upper right panel, a correct quadratic model is fitted (R^2=0.938). On the bottom left, a polynomial of degree 10 is fitted (R^2=0.945) and on the bottom right a polynomial of degree 15 is fitted (R^2=0.954). Even though the R^2 continues to improve, the higher degree polynomials (degree > 2) are overfit. Source: Nakatsu (2017).



To perform model validation, and find the λ that minimizes MSE, the most straightforward method is to randomly split a dataset into two sets: (1) a **training set** and (2) a **validation set**. A ridge regression model is built using the training set. The held-out validation set, which was not used to build the model, is used to validate the model. The validation process involves calculating the MSE over a range of λ values. The λ that results in the lowest MSE on the validation set is selected. Because the validation dataset is held-out, this method is sometimes referred to as the holdout method.

Unfortunately, the holdout method provides only a single estimate of a model's validation error, MSE. The split between training and validation sets could be a particularly biased choice—even if randomized—and could either underestimate or overestimate MSE. This could especially pose a problem when dealing with smaller datasets. A standard way to address this problem is to use **resampling**, which means repeatedly drawing randomized samples from a dataset and refitting the model on each sample [20]. By resampling, the average error rate of multiple runs can provide a less biased estimate of error rate than a single-point estimate could.

There are several approaches to resampling, but one of the most effective methods is **k-fold cross-validation (CV)**. This method was introduced in 1974 [2,13,35] and over the years has emerged as the most popular resampling method. Many practitioner guides and textbooks today advocate its use in model validation (see e.g., [1,12,14,20,24]). The method begins by randomly splitting a dataset into k partitions called folds (k = 5 or 10 folds is most commonly used, but other k sizes can be used as well). Subsequently, the technique iterates k times: on each iteration, one fold is set aside as the validation set, and the remaining k - 1 folds are used to train the model. The model thus built is validated only on the validation set. After iterating k times this way, an average of the k validation errors is calculated so that a more accurate and unbiased estimate of error can be obtained.

Repeated k-fold cross-validation (CV) is a method that can further improve the estimate of the validation error. Under this method, k-fold CV is repeated multiple times, and the average of the multiple repetitions is used to estimate the error rate. Given n repetitions, there will be n*k validation errors; hence, the average validation error is calculated over the n*k repetitions. The most common way of running k-fold CV is only once; thus, the repeated method has been suggested by others as a way of obtaining more accurate and reliable estimates of error rates [23,27]. In our own research lab, we have verified that repeated k-fold CV is the best general method for model validation on both classifiers [29] and regression models [30]. Because it has been shown empirically to produce the most accurate estimates of validation error, repeated k-fold CV is used in this study to understand model fit.

3 The Fitting Graph

Single estimates of model performance may be uninformative and lacking in context. On the other hand, presenting a large table of numbers would provide the needed context, but may be difficult to comprehend because the volume of data can be overwhelming to process and make sense out of. A solution is to create an information graphic, which can help the end-user gain insight into the data. In machine learning and data science, one information graphic is known as the fitting graph. This type of graphic shows model performance (predictive accuracy) as a function of model complexity [33]. Because level of complexity is the primary parameter to tweak to avoid model overfit and model underfit, a fitting graph is a natural choice for information visualization for many types of machine learning problems. However, the fitting graph has not been extensively studied in the literature in terms of how to estimate them—this is a dearth of information and practical guidelines on how to construct them directly from your data.

This study focuses exclusively on how fitting graphs can be used to support model development and validation and, thereby, result in a model selection process that is more trustworthy and explainable. Specifically, for ridge regression, the task involves the selection of the regularization parameter λ. The task can be summarized as this: How do you select λ, such that error, or MSE, in a ridge regression model is minimized?

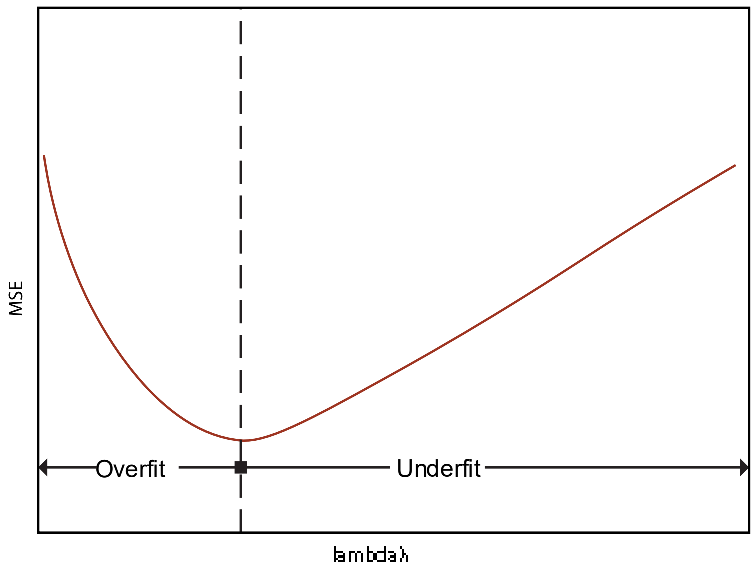
Fig. 2 illustrates a typical fitting graph that can be used to select the best-fitting λ in a ridge regression problem. This fitting graph exhibits a familiar u-shape: the MSE first declines as λ increases (representing model overfit), then reaches a minimum point, after which the MSE increases as λ increases (representing model underfit). The point at which the MSE is at a minimum is the best-fitting λ value.

If we had the correct fitting graph before us, then it would be a simple matter to read off the λ value where MSE is at a minimum. However, arriving at an accurate fitting graph can be difficult to achieve, as we will illustrate in the next section. Even when repeated k-fold CV is used—whose intent is to find more accurate estimates of MSE through multiple repetitions of k-fold CV—it can be difficult to construct an accurate fitting graph. In the practitioner literature, fitting graphs (like the one in Fig. 2) are shown more as "idealizations" of how error changes as a function of model complexity but provides no guidance on how to construct them or estimate them from your data.

The objectives of the study are to explore how to use the fitting graph as an explanations tool in model validation and address the following questions:

- Using a scatterplot of model complexity, as represented by λ, plotted against error, as represented by MSE, how do you know when repeated k-fold CV is choosing the best-fitting λ? That is, when can you be confident that you have performed enough iterations of k-fold CV?
- How do you estimate a fitting graph when there is significant noise in the data?
- How is a fitting graph used to support explainable and robust model development?

Fig. 2. A fitting graph for ridge regression modeling. A fitting graph has a typical u-shape (or inverted u-shape if you are looking at accuracy instead of error). This fitting graph shows error (MSE) as a function of model complexity, as determined by λ.



3.1 Constructing fitting graphs

To understand how to construct fitting graphs directly from your data, regression modeling is performed on two datasets.

1. **Baseball** [20]. Major League baseball data from the 1986 to 1987 seasons, n=263, p=19.
    Target variable: **Salary**
    Predictor variable examples:
    - Atbat (number of times at bat)
    - Hits (number of hits)
    - Runs (number of runs)
    - RBI (number of runs batted in)
    - Years (number of years in the major leagues)

2. **Boston** [17]. Housing values in the suburbs of Boston, n=506, p=13.
    Target variable: **Median Home Value**
    Predictor variable examples:
    - Crim (per capita crime rate by town)
    - Zn (proportion of residential land zoned for lots over 25,000 sq. ft.)
    - Rm (average number of rooms per dwelling)
    - Age (proportion of owner-occupied units built prior to 1940)
    - Dis (weighted mean of distances to five Boston employment centers)

Each modeling task involved the prediction of a numeric outcome (target variable) from a feature set (predictor variables). Both datasets were relatively small, with n (number of rows) and p (number of predictor variables) indicated above for each. Ridge regression was run on both datasets. The task involved determining which λ to select—i.e., which λ results in the lowest MSE. Although both datasets are small, it is shown in Appendix A how the same techniques will work on a larger dataset.

You will typically not know the size of λ beforehand. Finding a range of λ values to use will largely be a matter of trial and error. To begin your search, you should start out with widely spaced-out λ values. Then, based on your results, you can zoom in on the where the best values lay, using a narrower range of values. You might have to repeat this process a few times to find a suitable range of values. Through a few iterations of testing, the following ranges of λ were arrived at on the two datasets:

- On the baseball dataset, k-fold CV was run 100 times, for λ = 0.2 to 20, in 0.2 increments
- On the Boston dataset, k-fold CV was run 100 times, for λ = 0.01 to 1.00, in 0.01 increments

100 equally spaced λ values were used because the method of estimating the fitting graphs using regression splines (see discussion in Section 4) requires multiple λ values. 100 λ values turned out to be a good choice that allowed us to estimate fitting graphs directly from the data.

Once ridge regression was run on the 100 λ values, average MSE was plotted on a scatterplot over the 100 values of λ. Fig. 3 shows the results of k-fold CV for different levels of iterations (or reps) of k-fold CV on the baseball dataset. Fig.s 3 (a) through (d) show the scatterplots of MSE when plotted against different values of λ for (a) no reps, (b) 10 reps, (c) 1,000 reps and (d) 75,000 reps. The fitting graph curve can be more clearly discerned as the number of reps is increased. In Fig. 3 (a), when only single-run k-fold CV (no reps) is run, the scatterplot looks random, and it is hard to detect any trends in the data. In Fig. 3 (b), when 10 reps of k-fold CV is run, trends in the data are starting to emerge, but they are not clear. In Fig. 3 (c), using 1000 reps, the u-shape of a fitting graph can be discerned. Finally, Fig. 3 (d) shows that by 75,000 reps, there is almost a perfect u-shaped line formed by the data.

The same thing happens on the Boston dataset. Fig. 4 shows scatterplots on the Boston dataset for (a) no reps, (b) 10 reps, (c) 1,000 reps and (d) 10,000 reps. 10,000 reps were used on this dataset because it requied fewer iterations of k-fold CV to achieve a curved line, whereas 75,000 reps were required on the baseball dataset. This shows that there is no single number of iterations to perform to achieve convergence; rather, it depends on characteristics of your dataset, including the size of the dataset. Another difference with this dataset is that regularization barely improved the performance of the ridge regression model: λ = 0 (no regularization) was close to the minimum MSE, so the u-shape (or bend) in the curve is barely perceptible. Compare Fig. 3 (d) to Fig. 4 (d), where you can see the u-shape in the baseball dataset, but not in the Boston dataset

3.2 Explainable and robust model selection using the fitting graph

Earlier, we defined AI robustness broadly as AI that is reliable and resilent, performing accurately over a wide range of conditions. In the model development context, we want to look at the task of **tuning the hyperparameters** of a machine learning model (i.e., selecting the correct model). What does robustness mean in this context? Robustness can be defined as demonstrating insensitivity of a model's performance to variations in the hyperparameters [31,42]. That is, we expect the λ value that we choose will not be far off from the actual minimum. Because the estimation of λ is imprecise, subject to the noise in the data, we want to choose a λ value that is robust—that is, one that will not deteriorate too much (result in a low MSE) if we either overshoot or undershoot the actual optimum λ value. Let's illustrate how we can use the fitting graph to support an explainable and robust model selection process.

How can we be confident that we have chosen a λ value that is robust? Furthermore, how many reps of k-fold CV does it take to reach that level of confidence? Let's take a closer look at the scatterplots for the baseball dataset (Fig. 3) to address this question. As you can see from the four scatterplots, performing single-run (no reps) k-fold CV provides—see Fig. 3 (a)—provides little indication of which λ would result in the lowest MSE. By performing repeated k-fold CV with 10 reps—see Fig. 3 (b)—you can see that the lowest λ values result in higher MSEs, but it is still hard to discern the fitting graph curve in the scatterplot for larger values. It does appear, however, that λ values roughly from 2.5 to 20 all fall within the range of 115,000 to 117,500. If this is an acceptable range of variation in MSE, then you can accept any λ that falls within this range, and end your search at 10 reps. When the number of reps increases to 1,000—see Fig. 3 (c)—then you can fine-tune your choice of λ even further: λ values between 5 and 10 would all result in MSE values between 115,000 and 116,000. By 75,000 reps, the minimum MSE can be pinpointed almost precisely from the scatterplot: It occurs at λ = 6.6, which results in a minimum MSE = 115,569.

As this example illustrates, you can use the fitting graph plot to determine your level of confidence in the λ value: more reps will generate a more precise curve, but at the expense of higher computational costs. Moreover, the fitting graph will provide the range of values of your error (MSE). Even using the 10-rep plot—see Fig. 3 (b)—you can see that MSE falls between 115,000 and 118,000 for all λ values between 5 and 20. That result may be robust enough and achieve with few reps.

Fig. 3. Baseball dataset scatterplots: average MSE over a range of λ values.

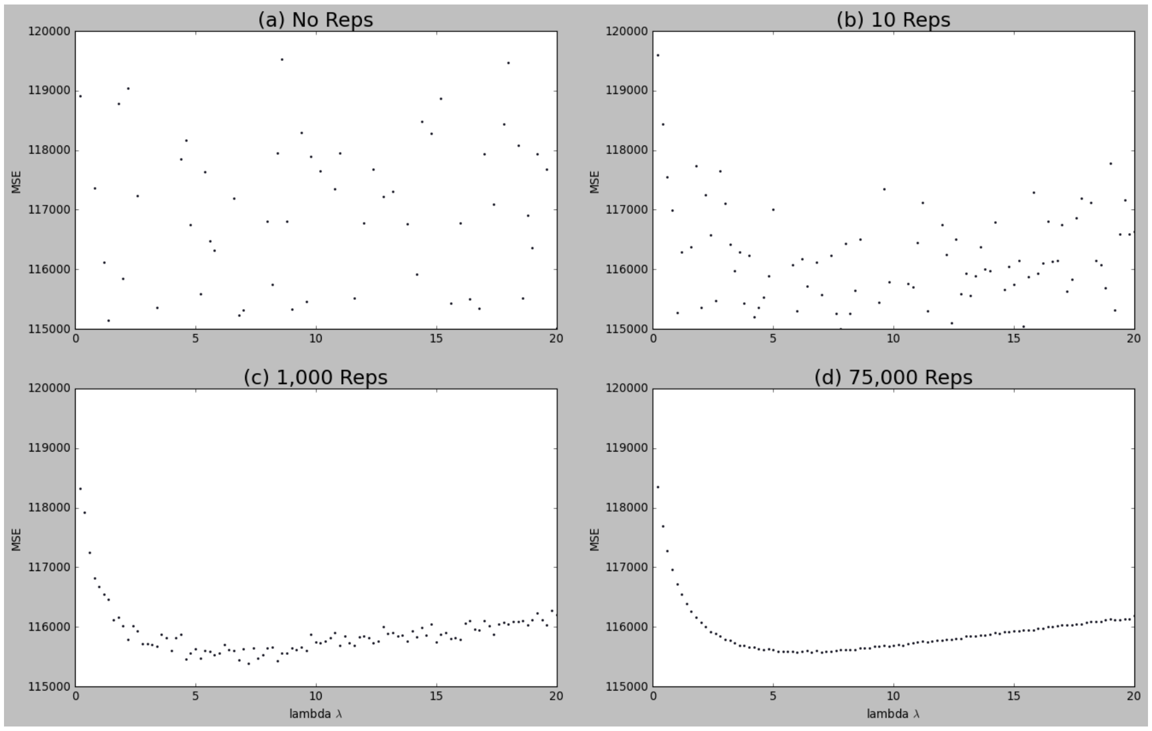
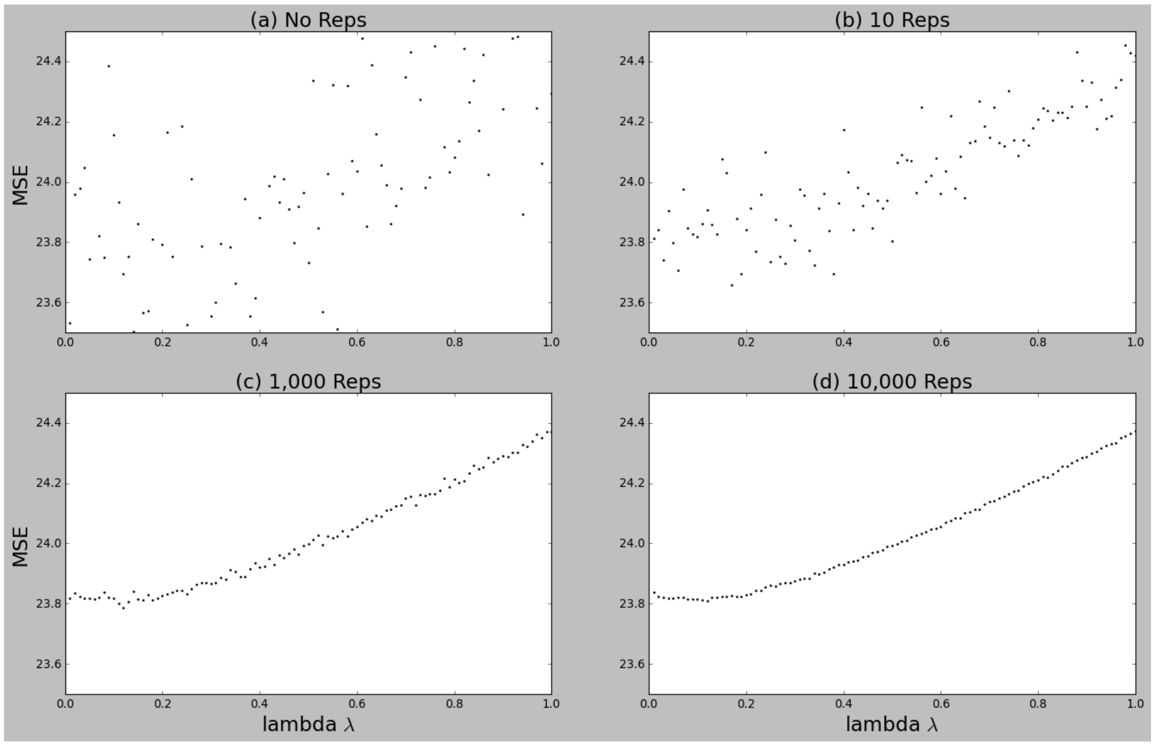


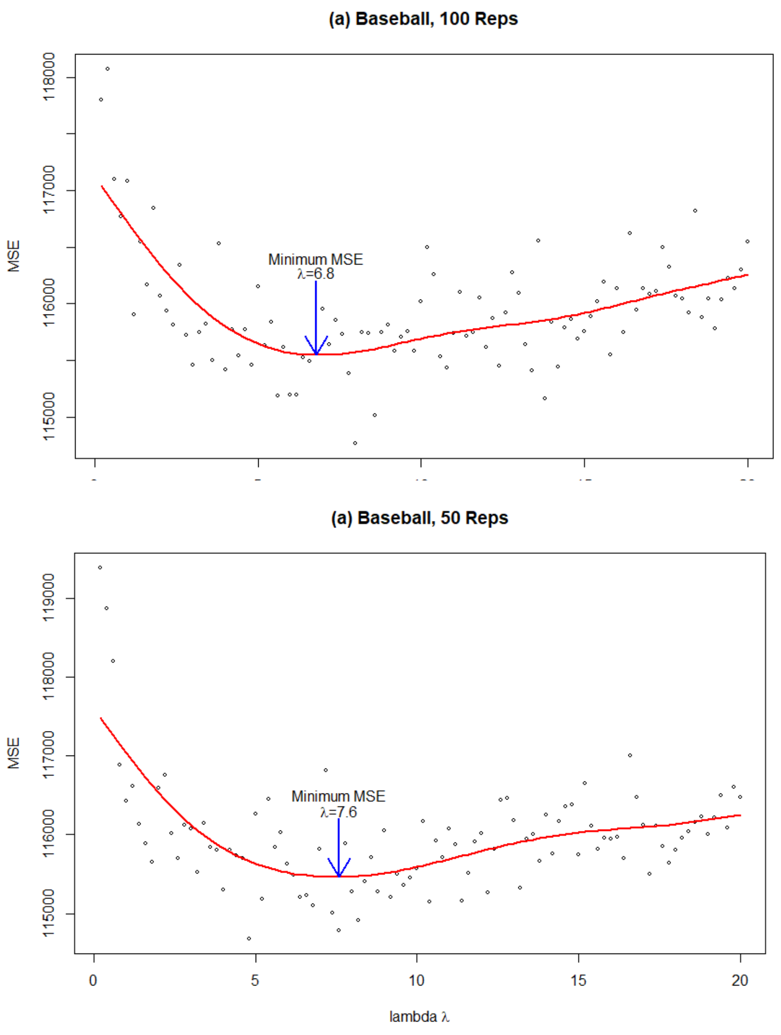
Fig. 4. Boston dataset scatterplots: average MSE over a range of λ values.



4 Regression Splines

75,000 reps of k-fold CV—and even 1,000 reps—would take a very long time to compute on all but the simplest of datasets. It is **not** the intent of this study to suggest that you should run k-fold CV a thousand times to estimate a fitting graph; rather, the large number of iterations was used to demonstrate that, given enough reps, you can generate a near-perfect curve. It was feasible enough to perform 1,000 iterations on a small dataset like the Baseball dataset (n=267, p=13), but running 1,000 reps may not be practical on larger datasets having large n (number of rows) and/or large p (number of features). In general, if many reps are required to generate an accurate fitting graph, it is recommended that you use a shortcut method instead: Derive the fitting graph curve using a regression spline. Through this technique, you can estimate the curve using a scatterplot generated on a much smaller number of reps.

Fig. 5. Regression splines are used to generate fitting graphs. The fitting graphs (the red curves) are estimated on (a) the baseball dataset using 100 reps and (b) the baseball dataset using 50 reps. Both fitting graphs do a good job in selecting a λ value.



Let's illustrate how this might be accomplished by using scatterplots for the baseball datasets, generated on 100 reps and 50 reps of k-fold CV. These scatterplots are shown in Fig. 5 (a) and Fig. 5 (b) for 100 reps and 50 reps, respectively. Fig. 5 shows the red curves, generated by regression splines, that are fit and overlaid on the scatterplots. We can then use these curves to estimate where the minimum MSE occurs. Based on the minimum on the regression spline curves, Fig. 5 (a) selects λ = 6.8, while Fig. 5 (b) selects λ = 7.6.

As noted previously the "actual" minimum MSE is located at λ=6.6 on the baseball dataset. To derive this value, the scatterplot generated from 75,000 reps of k-fold CV—refer to Fig. 3 (d)—was used to select the λ that results in the lowest MSE. These results show that the regression splines did a good job in estimating the minimum values, even when generated on as few as 50 reps. To verify these choices, look at the fitting graph obtained from Fig. 3 (d); you can see that λ values between 5 and 8 all represent good choices with a low MSE. Although 75,000 reps was needed to generate a smooth fitting graph curve directly from the scatterplot, 50 reps was sufficient to select a good value for λ, thus demonstrating how this technique can be computationally feasible on a far smaller number of reps.[3] In Appendix A, this result is replicated using a larger dataset where n=5,875; on this dataset only 10 reps are required to generate a reasonable fitting graph. In general, larger datasets will tend to require fewer reps to generate good fitting graphs. A rule-of-thumb recommendation is to use between 10 to 100 reps when estimating fitting graphs from your data.

Why use regression splines to estimate the fitting curves? Traditionally, the way to fit a non-linear curve onto a set of data points has been to replace the standard linear regression model with a polynomial function. This approach, known as polynomial regression, replaces the linear model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad (4)$$

with a polynomial function of the form

$$y_i = \beta_0 + \beta_1 x_i + \beta_1 x_i^2 + \beta_1 x_i^3 + \cdots + \beta_1 x_i^d + \epsilon_i \quad (5).$$

Where i is the ith observation in the dataset. For large values of d, polynomial regression will fit very flexible curves on your data. However, the problem with a high-order polynomial regression model is that it can take on highly irregular shapes and overfit your data as seen in Fig. 1. For this reason, it is generally not recommended to use a value of d greater than 3.

Regression splines typically perform better than polynomial regression, and result in better-fitting curves on your data. This is because regression splines involve fitting separate lower-degree polynomials (e.g., quadratic, or cubic polynomials) over different regions of X, whereas polynomial regression will fit a single higher-degree polynomial over the entire range of X [20]. By dividing the range of X into k distinct regions (k is known as the number of **knots**), you can fit a lower-degree polynomial on each region separately; hence, this approach is sometimes referred to as **piecewise polynomial regression**. Because you are fitting separate lower-degree polynomials on different sections of your

[3] The execution time for running 100 reps of k-fold CV over 100 λ values was 5 min 10 sec; and, for 50 reps, was half that, or 2 min 35 sec. These results were based on the execution time of running repeated k-fold CV on a computer equipped with an i7 Intel microprocessor. This shows how the generation of fitting graphs using repeated k-fold CV is computationally feasible for such datasets.

data, regression splines are less prone to overfitting. In addition to fitting separate polynomial functions on each region, the polynomial functions are constrained so that they join smoothly at the knots, or the boundaries between the separate regions.

How do you determine where the knots occur? One way is to place a knot where the estimated fitting graph curve would pivot or change its direction. For example, by looking at the scatterplot in Fig. 5 (a), you might create a knot somewhere in the data where the minimum point occurs—e.g., λ = 6.8. By doing so, you would create two separate polynomial regression models, over two separate ranges of λ: (1) λ < 6.8, and (2) λ >= 6.8. (You may not know exactly where the minimum occurs, but you can estimate the minimum based on the scatterplot). To avoid having to choose your pivot value, you can also specify a k value, in which case, k separate regions are automatically generated for you in a uniform way across your X values. For example, by specifying k = 4 on the baseball dataset, the software would divide your data into four uniformly sized regions:

```
Region 1:  0 < λ <=  5
Region 2:  5 < λ <= 10
Region 3: 10 < λ <= 15
Region 4: 15 < λ <= 20
```

There are libraries and functions in Python, R, and other programming environments that allow you to generate regression splines on your data. For example, in R, you can fit regression splines, using functions in the library **splines**, by either specifying the precise location of the knots, or by specifying k (number of knots); the latter specification would result in a regression spline with knots created at uniform quantiles of your data.

5 Case Study: Illustrating Robustness of the Fitting Graph Method on an Unstable Regression Model

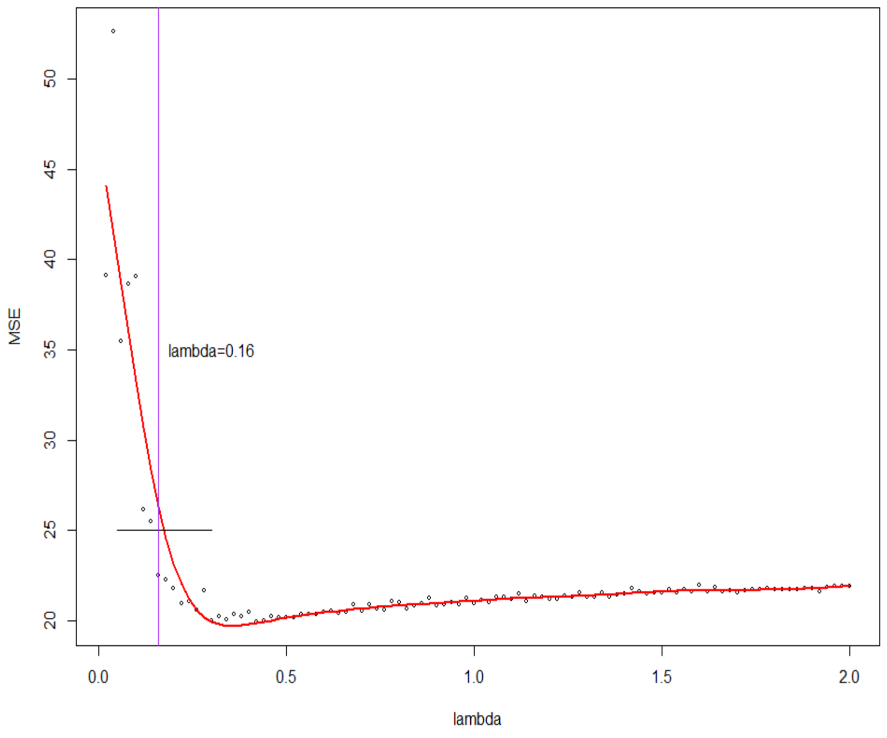
To generate a highly unstable ridge regression model, additional features were created from the Boston dataset, by raising the power of each of the original features up to a power of 8. That is, each of the 13 features in the Boston dataset was raised to a power from 1 through 8. Thus, for a feature x_i the features $x_i, x_i^2, x_i^3, x_i^4, \dots, x_i^8$ were created. Hence the number of features was expanded eight-fold to 13*8 or 104 features—henceforth, this dataset is referred to as the **Boston polynomial dataset**. The intent was to illustrate a ridge regression model that is highly unstable in its results. k-fold CV utterly fails in selecting the correct regression model. On the other hand, the fitting graph method is a more robust way of selecting a good λ value and can withstand the instability of the high-order polynomial model.

5.1 The fitting graph

Fig. 6 graphically shows the average MSE after 10-fold CV was repeated 40,000 times for 100 values of λ ranging from 0.02 to 2.0, in 0.02 increments. Due to the instability of the polynomial regression model, average MSE does not follow a smooth curve—even after 40,000 reps of 10-Fold CV—so a regression spline is used to estimate a fitting graph. See the red curve in Fig. 6.

Two things stand out in the fitting graph in Fig. 6. First, for values of λ < 0.16 (indicated by the vertical line in Fig. 6), MSEs were significantly higher than MSEs for the other λ values. You can see that MSE was above 25 for all these values and increased significantly as λ approaches 0. Second, for λ > 0.16, average MSE did not vary by that much, clustering around 20 – 22. It appears that choosing λ >= 0.16 and λ < 2.0 would result in reasonable and consistent results. In other words, robustness of the model is assured when choosing a λ value between 0.16 and 2.0.

Fig. 6. Average MSE on Boston polynomial dataset, 40,000 reps. The data points represent the average MSE over a range of 100 λ values (0.2 – 2.0) run 40,000 times on the Boston polynomial dataset. A regression spline is used to fit the red smooth curve to the data. For λ < 0.16, the average MSE becomes unstable and is at its highest. For all other λs average MSE is relatively stable, clustering in the 20 – 22 range.



This example demonstrates how a fitting graph, created with a regression spline, can be especially beneficial when the ridge regression results are highly unstable. Despite the average MSE not converging to a curved line—as the scatterplot in Fig. 6 illustrates—a fitting graph could easily be estimated using a regression spline.

5.2 Simulation study

The generation of a fitting graph can be important when there is instability in the ridge regression model. On the Boston polynomial dataset, a simulation study was conducted to investigate how k-fold CV would perform using either no iterations or a low number of iterations (1, 2, 5 or 10). In standard practice, this is usually how k-fold CV is run—either once or a few times. **Five variations** of k-fold CV were run:

1. 10-fold CV, single run
2. 5-fold CV, 2 reps
3. 50-fold CV, single run
4. 5-fold CV, 10 reps
5. 10-fold CV, 5 reps

Using the regression spline generated in Fig. 6, three categories of λ values were designated: (1) good fit (top 20% of λs); (2) average fit (middle 60% of λs) and (3) poor fit (bottom 20% of λs). These three categories were assigned based on the average MSE of the k-fold CV approach—the lower the MSE, the better the λ ranking. The MSEs were estimated from the fitting graph determined by a regression spline, as shown in Fig. 6.

Once fit types (good, average, poor) were assigned to the 100 λs, evaluation of the five variations could proceed as follows: For each of the five variations of CV, 1,000 runs were performed to see how often a good fit, average fit, and poor fit λ were chosen. For example, for 5-fold CV 10 reps, the Boston polynomial dataset was randomly split into five folds and the MSE was calculated five times, each time validated on a held-out fold. This was then repeated 10 times—each time on a randomized dataset—resulting in 5*10 or 50 MSE calculations. The average MSE was calculated over the 50 runs. This entire process was repeated on each of the 100 λs ranging from 0.02 to 2.0, in 0.02 increments. The λ that generated the lowest average MSE was selected. Table 1 presents the results of the simulation study.

The results are surprising: In all five variations of k-fold CV, a poor fit λ was selected by a wide margin.[4] The worst performers were 50-fold CV and 10-fold CV 5 reps, which both chose a poor fit λ 99.8% of the time! You do see some improvement when using 5-fold CV, 10 reps, which chose a poor fit λ 93% of the time, but no version of k-fold CV, either single-run or repeated, performed well.

How is it possible that k-fold CV performed so poorly, across the board, on the Boston polynomial dataset? To better understand why this happened, let's look more closely at the distribution of MSE over 1000 runs and compare two groups: (1) **Group I (Poorest Fit λs)** (.02,.04,.06,.08,.1)—the five λ values representing the poorest performance and (2) **Group II (Best Fit λs)** (.42,.44,.46,.48,.5)—the five λ values representing the best performance . The distribution of the 5,000 MSE values representing Group I λs (i.e., highest average MSEs) was compared to the 5,000 MSE values representing Group II λs (i.e., the lowest average MSEs). (Please note: There are a total of 5,000 values because there are five λs in each group). Table 2 shows the distributions of the two groups side-by-side. You can see that the distribution of Group I (poorest fit) is much more widely dispersed: it has more extreme values on both the low end as well as the high end (five values have MSE > 5000). By contrast, the distribution of Group

[4] In our research lab, when running k-fold CV and repeated k-fold CV on most other datasets, this did not happen. k-fold CV and repeated k-fold CV usually do a much better job of finding good fit λ values (Nakatsu, 2021; Nakatsu, 2023).

II (best fit) is much more concentrated: MSE is concentrated in the 18-20 range 85.6% of the time (4280 out of 5000). Average MSE of Group II is about half that of Group I (19.71 vs. 40.13) and the standard deviation is much lower (5.69 vs. 447.89). Hence, not only is bias higher but so is variance. Clearly, running ridge regression using a Group I λ (.02,.04,.06,.08,.1) resulted in a much more unstable ridge regression model.

Table 1. Simulation study results on five variations of k-fold CV, Boston polynomial dataset

	(1) 10-Fold CV	(2) 5-Fold CV 2 reps	(3) 50-Fold CV	(4) 5-Fold CV 10 reps	(5) 10-Fold CV 5 reps
Boston Polynomial Dataset					
Good Fit: Top 20%	8	18	1	24	0
Average Fit: Middle 80%	4	17	1	46	2
Poor Fit: Bottom 20%	988	965	998	930	998

Table 2: Distribution of MSE for Group I vs. Group II

Group I Poorest Fit λs	
Interval	**Frequency**
<=18	2179
(18-20]	2155
(20-22]	199
(22-24]	55
(24-26]	108
(26-100]	257
(100-500]	8
(500-5000]	16
(1000-5000]	18
>5000	5
Average	40.13
SD	447.87

Group II Best Fit λs	
Interval	**Frequency**
<=18	5
(18-20]	4280
(20-22]	687
(22-24]	5
(24-26]	0
(26-100]	21
(100-500]	2
(500-5000]	0
(1000-5000]	0
>5000	0
Average	19.71
SD	5.69

Yet, despite Group II having both lower bias and lower variance, why did k-fold CV (both single-run and repeated variations) end up, predominantly, choosing a Group I λ value instead? The answer is that Group I λ values were also unstable on the low side, generating a low-end MSE frequently as well (in Table 2, 2179 out of 5000 times). This means that k-fold CV was able to find, simply by chance, a low average MSE using a Group I λ—in effect choosing an inappropriate λ value that would result in an unstable model.

Varma and Simon [39] and Boulesteix and Strobl [3] point out that there is an inherent bias when estimating model performance (or error) over a large range of parameter values. Because the λ value associated with the smallest average validation error (i.e., MSE) is selected, this error is likely to be optimistic. It can be explained this way: When there are several possible λ values—in this study, 100 separate λ values are evaluated—the λ value that is finally chosen is likely to be optimistic because it is drawn from a random sample of possible errors. With several λ values to choose from, the resulting average MSE error is likely to be low simply by chance, rather than truly representing the lowest average MSE. This problem will especially be pronounced in unstable cases where there are many low-end extreme values, such as was reported in Table 2.

The result is that k-fold CV does not always work well, and you cannot always trust its results. In this example, the failure of k-fold CV is clear and unmistakable—in some cases an unstable λ would be selected close to 99% of the time. In situations like these, the fitting graph can provide the model developer with a more complete picture—more explanation about what is happening, and what to do instead. Without the fitting graph, the model designer of an unstable model is likely to be led astray. Clearly, in this example, the fitting graph method led to a more robust selection of a λ value.

6 Discussion and Conclusions

In the practitioner literature, the standard recommendation for model validation is to perform k-fold CV—most typically using 5 or 10 folds [22]. In most cases, the model designer will perform this only once, rather than performing repeated k-fold CV. In many cases, this k-fold CV procedure works fine, and will produce reasonable results. However, this study demonstrates how k-fold CV (whether single-run or repeated 5 - 10 times) may not provide enough accuracy in your results; moreover, the robustness of the model selected might be lacking. In some cases, more explanation may be required to understand whether the correct model has been selected.

This research demonstrates how fitting graphs can enhance explainability for the model designer, leading to greater trust in the model selection process. In this study, the fitting graph was actively constructed directly from the data and was used as a tool to support model validation, not merely created as a static rendering of how accuracy changes as a function of model complexity, which is how the current literature treats the fitting graph (see e.g., [3]). We know of no prior studies that have used the fitting graph in this way or have demonstrated how to create fitting graphs directly from your data. The model validation process was illustrated using a ridge regression task: Find the best-fitting regularization parameter λ among a range of λ values. First, it was demonstrated how a simple scatterplot of the λ values (representing model complexity) plotted against average MSE (representing predictive accuracy) would provide the model designer with an information graphic on whether enough iterations of k-fold CV have been performed. A highly erratic scatterplot may indicate more iterations might be necessary whereas a plot in which a u-shaped curve can be discerned may indicate that enough iterations have been performed. The scatterplots, alone, provide a useful explanatory visualization on whether the model designer is on the right track. Second, fitting curves were constructed from noisy scatterplots using regression splines. Rather than increasing the number of iterations of k-fold CV, which would require more computational resources to perform, regression splines can save you time in estimating the fitting graphs. One benefit of regression splines is that they require very little computational power to generate and are easy to compute using standard software routines readily available in R, Python, and other popular machine learning libraries.

Furthermore, model robustness is addressed in this study. The type of model robustness that was addressed involves the selection of the hyperparameter λ when designing a ridge regression model. As Nobandegani et al. [31] state, a robust model is one that is performance insensitive to variations in the parameters (i.e., hyperparameters). In Section 3.2, we illustrate these ideas by showing how the fitting graph can help us choose a λ value that is insensitive to regression model performance, or MSE. Secondly, we illustrate model robustness by showing how k-fold CV can fail spectacularly on an unstable ridge regression model. Again, the solution was to generate (or estimate) the fitting graph from a regression spline on a scatterplot of your data. For the polynomial regression model described in Section 5, k-fold CV ended up choosing a poorly fit and unstable λ by an overwhelming margin. By applying the fitting graph method, the model developer can choose a better-fitting and more robust λ. Hence, fitting graphs not only reassure the model developer that they are on the right track—enhancing the trustworthiness of the model-building process—but also guide them toward better models. In ridge regression, for example, this can involve selecting a more stable and well-fitting λ value.

Although this research deals exclusively with regression modeling, and the tuning of the regularization parameter λ, these principles and techniques can generalize to other machine learning algorithms. The fitting graph method can be extended to other popular machine learning algorithms, including logistic regression, random forests, support vector machines, and neural networks. In all cases the basic application of the fitting graph remains the same: the curve provides a visualization on how error (or predictive accuracy) changes as a function of model complexity. Future work will look at how these concepts can be extended to create trustworthy AI for these other machine learning methods.

REFERENCES

1. Y.S. Abu-Mostafa, M. Magdon-Ismail and H.T. Lin, *Learning from data: A short course*, AMLBook, New York, 2012.

2. D.M. Allen, The relationship between variable selection and data augmentation and a method for prediction, *Technometrics* 16(1) (1974), 125–127.

3. A.L. Boulesteix and C. Strobl, Optimal classifier selection and negative bias in error rate estimation: An empirical study on high-dimensional prediction, *BMC Medical Research Methodology* 9(1) (2009), 1–14.

4. H.B. Braiek and F. Khomh, Machine learning robustness: A primer, *arXiv preprint* arXiv:2404.00897, 2024.

5. B.G. Buchanan and E.H. Shortliffe, *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA, 1984.

6. N. Burkart and M.F. Huber, A survey on the explainability of supervised machine learning, *Journal of Artificial Intelligence Research* 70 (2021), 245–317.

7. B. Chandrasekaran, M.C. Tanner and J.R. Josephson, Explaining control strategies in problem solving, *IEEE Expert Systems* 4(1) (1989), 9–24.

8. P. Cortez and M.J. Embrechts, Opening black box data mining models using sensitivity analysis, in: *Computational Intelligence and Data Mining (CIDM)*, IEEE, Piscataway, NJ, 2011, pp. 341–348.

9. T. Decker, R. Gross, A. Koebler, M. Lebacher, R. Schnitzer and S.H. Weber, The thousand faces of explainable AI along the machine learning life cycle: Industrial reality and current state of research, in: *International Conference on Human-Computer Interaction*, Springer, Cham, 2023, pp. 184–208.

10. S. Dhanorkar, C.T. Wolf, K. Qian, A. Xu, L. Popa and Y. Li, Who needs to know what, when?: Broadening the explainable AI (XAI) design space by looking at explanations across the AI lifecycle, in: *Designing Interactive Systems Conference*, ACM, New York, 2021, pp. 1591–1602.

11. European Commission, *Ethics guidelines for trustworthy AI*, 2019. Available: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai (accessed 18 September 2024).

12. P. Flach, *Machine learning: The art and science of algorithms that make sense of data*, Cambridge University Press, Cambridge, 2012.

13. S. Geisser, The predictive sample reuse method with applications, *Journal of the American Statistical Association* 70(350) (1975), 320–328.

14. A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, Sebastopol, CA, 2019.

15. A. Goldstein, A. Kapelner, J. Bleich and E. Pitkin, Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation, *Journal of Computational and Graphical Statistics* 24(1) (2015), 44–65.

16. I. Goodfellow, Y. Bengio and A. Courville, Regularization for deep learning, in: *Deep learning*, MIT Press, Cambridge, MA, 2016, pp. 216–261.

17. D. Harrison Jr. and D.L. Rubinfeld, Hedonic housing prices and the demand for clean air, *Journal of Environmental Economics and Management* 5(1) (1978), 81–102.

18. D. Hendrycks and T.G. Dietterich, Reliable machine learning, *Communications of the ACM* 64(10) (2019), 45–53.

19. A.E. Hoerl and R.W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, *Technometrics* 12(1) (1970), 55–67.

20. G. James, D. Witten, T. Hastie and R. Tibshirani, *An introduction to statistical learning*, Springer, New York, 2013.

21. D. Kaur, S. Uslu, K.J. Rittichier and A. Durresi, Trustworthy artificial intelligence: A review, *ACM Computing Surveys* 55(2) (2022), 1–38.

22. R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, Vol. 2, pp. 1137–1145.

23. M. Kuhn and K. Johnson, *Applied predictive modeling*, Springer, New York, 2013.

24. B. Lantz, *Machine learning with R: Expert techniques for predictive modeling*, Packt Publishing, Birmingham, UK, 2019.

25. B. Li, P. Qi, B. Liu, S. Di, J. Liu, J. Pei et al., Trustworthy AI: From principles to practices, *ACM Computing Surveys* 55(9) (2023), 1–46.

26. D.W. Marquardt, Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation, *Technometrics* 12(3) (1970), 591–612.

27. A.M. Molinaro, R. Simon and R.M. Pfeiffer, Prediction error estimation: A comparison of resampling methods, *Bioinformatics* 21(15) (2005), 3301–3307.

28. R.T. Nakatsu, Information visualizations used to avoid the problem of overfitting in supervised machine learning, in: *International Conference on HCI in Business, Government, and Organizations*, Springer, Cham, 2017, pp. 373–385.

29. R.T. Nakatsu, An evaluation of four resampling methods used in machine learning classification, *IEEE Intelligent Systems* 36(3) (2021), 51–57.

30. R.T. Nakatsu, The validation of machine learning ridge regression models using Monte Carlo, bootstrap, and variations of cross-validation, *Journal of Intelligent Systems* 32(1) (2023), 1–19.

31. A.S. Nobandegani, K. da Silva Castanheira, T. O'Donnell and T.R. Shultz, On robustness: An undervalued dimension of human rationality, in: *Proceedings of the 41st Annual Meeting of the Cognitive Science Society (CogSci)*, 2019, p. 3327.

32. OECD, *OECD AI principles*, 2019. Available at: https://oecd.ai/en/ai-principles (accessed 9 September 2024).

33. F. Provost and T. Fawcett, *Data science for business: What you need to know about data mining and data-analytic thinking*, O'Reilly Media, Sebastopol, CA, 2013.

34. C. Song, K. He, L. Wang and J.E. Hopcroft, Towards robust neural networks via random self-ensemble, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, Vol. 33, pp. 5491–5498.

35. M. Stone, Cross-validatory choice and assessment of statistical predictions, *Journal of the Royal Statistical Society: Series B (Methodological)* 36(2) (1974), 111–133.

36. M. Subianto and A. Siebes, Understanding discrete classifiers with a case study in gene prediction, in: *Proceedings of the Seventh IEEE International Conference on Data Mining*, IEEE,

Piscataway, NJ, 2007, pp. 661–666.

37. W.R. Swartout, XPLAIN: A system for creating and explaining expert consulting programs, *Artificial Intelligence* 21(3) (1983), 285–325.

38. A. Tsanas, M.A. Little, P. McSharry and L. Ramig, Accurate telemonitoring of Parkinson’s disease progression by noninvasive speech tests, *IEEE Transactions on Biomedical Engineering* 57(4) (2009), 884–893.

39. S. Varma and R. Simon, Bias in error estimation when using cross-validation for model selection, *BMC Bioinformatics* 7(1) (2006), 1–8.

40. K.R. Varshney, *Trustworthy machine learning*, Independently Published, Chappaqua, NY, 2022.

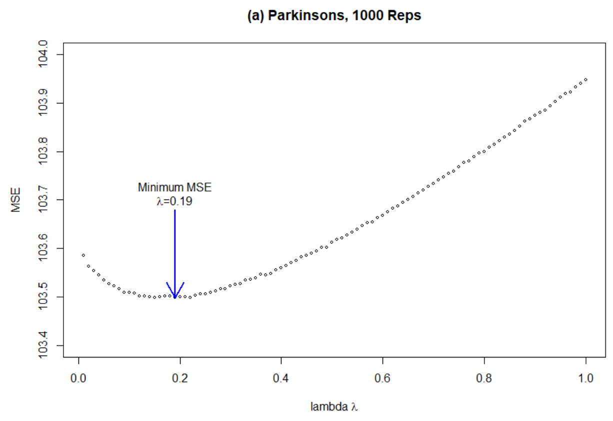
41. A. Wong, E. Otles, J.P. Donnelly, A. Krumm, J. McCullough, O. DeTroyer-Cooley et al., External validation of a widely implemented proprietary sepsis prediction model in hospitalized patients, *JAMA Internal Medicine* 181(8) (2021), 1065–1070.

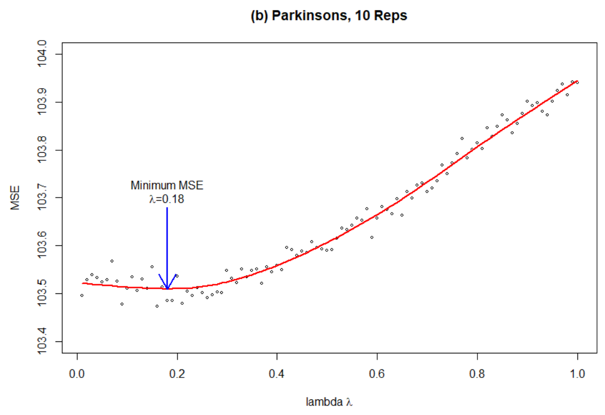
42. C. Zhang, A. Liu, X. Liu, Y. Xu, H. Yu, Y. Ma and T. Li, Interpreting and improving adversarial robustness of deep neural networks with neuron sensitivity, *IEEE Transactions on Image Processing* 30 (2020), 1291–1304.

Appendix A: Verification of Results Using Parkinsons Dataset

The Parkinsons dataset [38] contains biomedical voice measurements from 42 people with early-stage Parkinson's disease (n=5875, p=17 voice measurements). The target variable is Total-UPDRS, which stands for Unified Parkinson's Disease Rating Scale. Ridge regression was used to fit models on this dataset, using 100 λ values from 0.1 to 1.0. Fig. 7 (a) shows the results of MSE plotted against λ on 1000 reps of k-fold CV. You can see that with a large number of reps, the scatterplot approaches a near-perfect fitting graph curve with the familiar u-shape. Fig. 7 (b) shows the scatterplot generated on 10 reps of k-fold CV. The fitting graph is estimated using a regression spline, as represented by the red curve that is overlaid on the data. Using the regression spline, λ = 0.18 is chosen as the best-fitting λ having the lowest MSE. The actual minimum value, from Fig. 7 (a) is λ = 0.19. Again, the regression spline does a very good job in selecting a good value, using only 10 reps of k-fold CV.

Fig. 7. Results on the Parkinsons dataset (n=5875, p=17). (a) The scatterplot is generated on 1000 reps of k-fold CV. (b) The scatterplot is generated on 10 reps of k-fold CV, with the fitting graph (red curve) estimated by a regression spline.


(a) Parkinsons, 1000 Reps
MSE
Minimum MSE
λ=0.19
lambda λ


(b) Parkinsons, 10 Reps
MSE
Minimum MSE
λ=0.18
lambda λ