OXFORD

(GIGA)n SCIENCE

GigaScience, 2017, 1–12

doi: xx.xxxx/xxxx
Manuscript in Preparation
Paper

PAPER

Fractional Ridge Regression: a Fast, Interpretable Reparameterization of Ridge Regression

Ariel Rokem[1,*] and Kendrick Kay[2,*]

[1]Department of Psychology and the eScience Institute, University of Washington, Seattle, WA and [2]Center for Magnetic Resonance Research, University of Minnesota, Twin Cities, MN

[*]arokem@uw.edu; kay@umn.edu

Abstract

Background: Ridge regression is a regularization technique that penalizes the L2-norm of the coefficients in linear regression. One of the challenges of using ridge regression is the need to set a hyperparameter (α) that controls the amount of regularization. Cross-validation is typically used to select the best α from a set of candidates. However, efficient and appropriate selection of α can be challenging. This becomes prohibitive when large amounts of data are analyzed. Because the selected α depends on the scale of the data and correlations across predictors, it is also not straightforwardly interpretable.
Results: The present work addresses these challenges through a novel approach to ridge regression. We propose to reparameterize ridge regression in terms of the ratio γ between the L2-norms of the regularized and unregularized coefficients. We provide an algorithm that efficiently implements this approach, called fractional ridge regression, as well as open-source software implementations in Python and MATLAB (https://github.com/nrdg/fracridge). We show that the proposed method is fast and scalable for large-scale data problems. In brain imaging data, we demonstrate that this approach delivers results that are straightforward to interpret and compare across models and datasets.
Conclusion: Fractional ridge regression has several benefits: the solutions obtained for different γ are guaranteed to vary, guarding against wasted calculations, and automatically span the relevant range of regularization, avoiding the need for arduous manual exploration. These properties make fractional ridge regression particularly suitable for analysis of large complex datasets.

Key words: Generalized linear model; Hyperparameters; Brain imaging; Open-source software

Introduction

Consider the standard linear model setting $Y = X\beta$ solved for β, where Y is a data matrix of dimensionality d by t (d data points in each of t targets), X is the design matrix with dimensionality d by p (d data points for each of p predictors), and β is a coefficient matrix with dimensionality p by t (with p coefficients, one for each predictor, for each of the targets). Ordinary least-squares regression (OLS) and regression based on the Moore-Penrose pseudoinverse (in cases where $p > d$) attempt to find the set of coefficients β that minimize squared error for each of the targets y. While these unregularized approaches have some desirable properties, in practical applications where noise is present, they tend to overfit the coefficient parameters to the noise present in the data. Moreover, they tend to cause unstable parameter estimates in situations where predictors are highly correlated.

Ridge regression [1] addresses these issues by trading off the addition of some bias for the reduction of eventual error (e.g., measured using cross-validation [2, 3]). It does so by penalizing not only the sum of the squared errors in fitting the data for each target, but by also minimizing the squared L2-norm of the solution, $||\beta||_2^2 = \sum(\beta^2)$. Fortunately, this form of regularization does not incur a substantial computational

Compiled on: October 20, 2020.
Draft manuscript prepared by the author.

Key Points

- Ridge regression is a powerful and popular technique for regularizing linear regression, but finding the optimal degree of regularization can be challenging, particularly in large datasets.
- We propose a technique, fractional ridge regression, that reparameterizes ridge regression in terms of the ratio between the L2-norms of the regularized and unregularized coefficients.
- Fractional ridge regression is fast and scalable for large-scale data problems and delivers results that are straightforward to interpret and compare across models and datasets.

cost. This is because it can be implemented using the same numerical approach for solving unregularized regression, with the simple addition of a diagonal matrix αI to the standard matrix equations. Thus, the computational cost of solving ridge regression is essentially identical to that of the unregularized solution. Thanks to its simplicity, computational expedience, and its robustness in different data regimes, ridge regression is a very popular technique, with the classical references describing the method [1, 4] cited more than 25,000 times according to Google Scholar.

However, beneath the apparent simplicity of ridge regression is the fact that for most applications, it is impossible to determine *a priori* the degree of regularization that yields the best solution. This means that in typical practice, researchers must test several different hyperparameter values α and select the one that yields the least cross-validation error on a set of data specifically held out for hyperparameter selection. In large-scale data problems, the number of data points d, number of predictors p, and/or number of targets t can be quite large. This has the consequence that the number of hyperparameter values that are tested, f, can pose a prohibitive computational barrier.

Given the difficulty of predicting the effect of α on solution outcomes, it is common practice to test values that are widely distributed on a log scale (for example, see [5]). Although this approach is not grounded in a particular theory, as long as the values span a large enough range and are spaced densely enough, an approximate minimum of the cross-validation error is likely to be found. But testing many α values can be quite costly, and the practitioner might feel tempted to cull the set of values tested. In addition, it is always a possibility that the initial chosen range might be mismatched to the problem at hand. Sampling α values that are too high or too low will produce non-informative candidate solutions that are either over-regularized (α too high) or too similar to the unregularized solution (α too low). Thus, in practice, conventional implementations of ridge regression may produce poor solutions and/or waste substantial computational time.

Here, we propose a simple reparameterization of ridge regression that overcomes the aforementioned challenges. Our approach is to produce coefficient solutions that have an L2-norm that is a pre-specified fraction of the L2-norm of the unregularized solution. In this approach, called *fractional ridge regression* (FRR), redundancies in candidate solutions are avoided because solutions with different fractional L2-norms are guaranteed to be different. Moreover, by targeting fractional L2-norms that span the full range from 0 to 1, the FRR approach explores the full range of effects of regularization on β values from under- to over-regularization, thus assuring that the best possible solution is within the range of solutions explored. We provide a fast and automated algorithm to calculate FRR, and provide open-source software implementations in Python and MATLAB. We demonstrate in benchmarking simulations that FRR is computationally efficient for even extremely large data problems, and we show that FRR applies successfully to real-world data and delivers clear and interpretable results. Overall, FRR may prove particularly useful for researchers tackling large-scale datasets where automation, efficiency, and interpretability are critical.

Methods

Background and theory

Consider the dataset Y with dimensionality d (number of data points) by t (number of targets). Each column in Y represents a separate target for linear regression:

$$y = X\beta + \epsilon \tag{1}$$

where y is the measured data for a single target (dimensionality d by 1), X is the "design" matrix with predictors (dimensionality d by p), β are the coefficients (dimensionality p by 1), and ϵ is a noise term. Our typical objective is to solve for β in a way that minimizes the squared error. If X is full rank, the ordinary least squares (OLS) solution to this problem is:

$$\hat{\beta}^{OLS} = (X^\intercal X)^{-1} X^\intercal y, \tag{2}$$

where $X^\intercal$ is the transpose of X. This solution optimally finds the values of β that provide the minimal sum-of-squared error on the data: $\sum (y - X\beta)^2$. In cases where X is not full rank, the OLS solution is no longer well-defined and the Moore-Penrose pseudoinverse is used instead. We will refer to these unregularized approaches collectively as OLS.

To regularize the OLS solution, ridge regression applies a penalty (α) to the squared L2-norm of the coefficients, leading to a different estimator for β:

$$\hat{\beta}^{RR} = (X^\intercal X + \alpha I)^{-1} X^\intercal y \tag{3}$$

where α is a hyperparameter and I is the identity matrix [1, 4]. For computational efficiency, it is well known that the original problem can be rewritten using singular value decomposition (SVD) of the matrix X [6]:

$$X = USV^\intercal \tag{4}$$

with U having dimensionality d by p, S having dimensionality p by p, and V having dimensionality p by p.

Note that S is a square matrix:

$$S = \begin{bmatrix} \lambda_1 & 0 & \dots & & \\ 0 & \lambda_2 & 0 & \dots & \\ 0 & 0 & \lambda_3 & 0 & \dots \\ \vdots & & & & \\ \dots & 0 & 0 & 0 & \lambda_p \end{bmatrix}$$

with λ_i as the singular values ordered from largest to smallest. Replacing the design matrix X with its SVD, we obtain:

$$y = USV^\intercal \beta + \epsilon. \tag{5}$$

Given that U and V are unitary (e.g., $U^\intercal U$ is I), left-multiplying each side with $U^\intercal$ produces:

$$U^\intercal y = SV^\intercal \beta + U^\intercal \epsilon. \tag{6}$$

Let $\tilde{y} = U^t y$, $\tilde{\beta} = V^\intercal \beta$, and $\tilde{\epsilon} = U^t \epsilon$. These are transformations (rotations) of the original quantities (y, β, and ϵ) through the unitary matrices U^t and V^t. In cases where $p < d$, this also projects the quantities into a lower-dimensional space of dimensionality p. The OLS solution can be obtained in this space:

$$\tilde{\hat{\beta}}^{OLS} = (S^\intercal S)^{-1} S^\intercal \tilde{y}, \tag{7}$$

which simplifies to:

$$\tilde{\hat{\beta}}^{OLS} = S^{-2}(S^\intercal \tilde{y}), \tag{8}$$

where

$$S^{-2} = \begin{bmatrix} \frac{1}{\lambda_1^2} & 0 & \dots & & \\ 0 & \frac{1}{\lambda_2^2} & 0 & \dots & \\ 0 & 0 & \frac{1}{\lambda_3^2} & 0 & \dots \\ \vdots & & & & \\ \dots & 0 & 0 & 0 & \frac{1}{\lambda_p^2} \end{bmatrix}$$

is the inverse of the square of the singular value matrix S. Thus, for a single coordinate i in the lower-dimensional space, we can solve the OLS problem with a scalar multiplication:

$$\tilde{\hat{\beta}}_i^{OLS} = \frac{1}{\lambda_i^2} \lambda_i \tilde{y}_i, \tag{9}$$

which simplifies finally to

$$\tilde{\hat{\beta}}_i^{OLS} = \frac{\tilde{y}_i}{\lambda_i}. \tag{10}$$

The SVD-based reformulation of regression described above is additionally useful as it provides insight into the nature of ridge regression [7]. Specifically, consider the ridge regression solution in the low-dimensional space:

$$\tilde{\hat{\beta}}^{RR} = (S^\intercal S + \alpha I)^{-1} S^\intercal \tilde{y} \tag{11}$$

To compute this solution, we note that:

$$S^t S + \alpha I = \begin{bmatrix} \lambda_1^2 + \alpha & 0 & \dots & & \\ 0 & \lambda_2^2 + \alpha & 0 & \dots & \\ 0 & 0 & \lambda_3^2 + \alpha & 0 & \dots \\ \vdots & & & & \\ \dots & 0 & 0 & 0 & \lambda_p^2 + \alpha \end{bmatrix} \tag{12}$$

the inverse of which is:

$$(S^t S + \alpha I)^{-1} = \begin{bmatrix} \frac{1}{\lambda_1^2+\alpha} & 0 & \dots & & \\ 0 & \frac{1}{\lambda_2^2+\alpha} & 0 & \dots & \\ 0 & 0 & \frac{1}{\lambda_3^2+\alpha} & 0 & \dots \\ \vdots & & & & \\ \dots & 0 & 0 & 0 & \frac{1}{\lambda_p^2+\alpha} \end{bmatrix} \tag{13}$$

Finally, plugging into equation 11, we obtain:

$$\tilde{\hat{\beta}}_i^{RR} = \frac{\lambda_i}{\lambda_i^2 + \alpha} \tilde{y}_i \tag{14}$$

This shows that in the low-dimensional space, ridge regression can be solved using scalar operations.

To further illustrate the relationship between the ridge regression and OLS solutions, by plugging equation 10 into equation 14, we observe the following:

$$\tilde{\hat{\beta}}_i^{RR} = \frac{\lambda_i^2}{\lambda_i^2 + \alpha} \tilde{\hat{\beta}}_i^{OLS} \tag{15}$$

In other words, the ridge regression coefficients are simply scaled-down versions of the OLS coefficients, with a different amount of shrinkage for each coefficient. Coefficients associated with larger singular values are less shrunken than those with smaller singular values.

To obtain solutions in the original space, we left-multiply the coefficients with V:

$$\hat{\beta} = V \tilde{\hat{\beta}} \tag{16}$$

We now turn to fractional ridge regression (FRR). The core concept of FRR is to reparameterize ridge regression in terms of the amount of shrinkage applied to the overall L2-norm of the solution. Specifically, we define the fraction γ as:

$$\gamma = \frac{||\tilde{\hat{\beta}}^{RR}||_2}{||\tilde{\hat{\beta}}^{OLS}||_2} \tag{17}$$

Because V is a unitary transformation, the L2-norm of a coefficient solution in the low-dimensional space, $||\tilde{\hat{\beta}}||_2$, is identical to the L2-norm of the coefficient solution in the original space, $||\hat{\beta}||_2$. Thus, we can operate fully within the low-dimensional space and be guaranteed that the fractions will be maintained in the original space.

In FRR, instead of specifying desired values for α, we instead specify values of γ between 1 (no regularization) and 0 (full regularization, corresponding to shrinking all the coefficients to $\beta = 0$). But how can one compute the ridge regression solution for a specific desired value of γ? Based on equations 9

and 14, it is easy to calculate the value of γ corresponding to a specific given α value:

$$\gamma = \frac{||\tilde{\hat{\beta}}^{RR}||_2}{||\tilde{\hat{\beta}}^{OLS}||_2} = \sqrt{\frac{\sum (\frac{\lambda_i \tilde{y}_i}{\lambda_i^2+\alpha})^2}{\sum (\frac{\tilde{y}_i}{\lambda_i})^2}} \quad (18)$$

In some special cases, this calculation can be considerably simplified. For example, if the singular value spectrum of X is flat ($\lambda_i = \lambda_j$ for any $i \neq j$), we can set all the singular values to λ, yielding the following:

$$\gamma = \sqrt{\frac{(\frac{\lambda}{\lambda^2+\alpha})^2 \sum \tilde{y}_i^2}{(\frac{1}{\lambda})^2 \sum \tilde{y}_i^2}} = \frac{\frac{\lambda}{\lambda^2+\alpha}}{\frac{1}{\lambda}} = \frac{\lambda^2}{\lambda^2+\alpha}, \quad (19)$$

This recapitulates the result obtained in [1], equation 2.6. We can then solve for α:

$$\alpha = \lambda^2(\frac{1}{\gamma} - 1) \quad (20)$$

Thus, in this case, there is an analytic solution for the appropriate α value, and one can proceed to compute the ridge regression solution using equation 14.

Another special case is if we assume that the absolute values of $\tilde{\beta}_i^{OLS}$ are all the same. In this case, we can use a few simplifications to calculate the shrinkage in terms of L1-norm:

$$\begin{aligned} \frac{||\tilde{\hat{\beta}}^{RR}||_1}{||\tilde{\hat{\beta}}^{OLS}||_1} &= \frac{\sum \left| \frac{\lambda_i^2 \tilde{\beta}_i^{OLS}}{\lambda_i^2+\alpha} \right|}{\sum \left| \tilde{\beta}_i^{OLS} \right|} \\ &= \frac{\sum \left| \frac{\lambda_i^2 \frac{\tilde{y}_i}{\lambda_i}}{\lambda_i^2+\alpha} \right|}{\sum \left| \frac{\tilde{y}_i}{\lambda_i} \right|} = \frac{\sum \frac{\lambda_i^2 \left| \frac{\tilde{y}_i}{\lambda_i} \right|}{\lambda_i^2+\alpha}}{\sum \left| \frac{\tilde{y}_i}{\lambda_i} \right|} \\ &= \frac{\sum \frac{\lambda_i^2}{\lambda_i^2+\alpha}}{p} \end{aligned} \quad (21)$$

Notice that this is the average of the shrinkages for individual coefficients from equation 15. The sum of these shrinkages (this quantity multiplied by p):

$$\sum \frac{\lambda_i^2}{\lambda_i^2+\alpha} \quad (22)$$

has been defined as the *effective degrees of freedom* of ridge regression (See [8], pg. 68). Note that the L1-norm here refers to the rotated space and may not be identical to the L1-norm in the original space.

These two special cases have the appealing feature that the regularization level can be controlled on the basis of examining only the design matrix X. However, they rely on strong assumptions that are not guaranteed to hold in general. Thus, for accurate ridge regression outcomes, we see no choice but to develop an algorithm that uses both the design matrix X and the data values y.

Algorithm

Our proposed algorithm for solving FRR is straightforward: it evaluates γ for a range of α values and uses interpolation to determine the α value that achieves the desired fraction γ. Although this method relies on brute force and may not seem mathematically elegant, it achieves accurate outcomes and, somewhat surprisingly, can be carried out with minimal computational cost.

The algorithm receives as input a design matrix X, target variables Y, and a set of requested fractions γ. The algorithm calculates the FRR solutions for all targets in Y, returning estimates of the coefficients $\hat{\beta}$ as well as the values of hyperparameter α that correspond to each requested γ. In the text below, we indicate the lines of code that implement each step of the algorithm (see also section Software implementation below) in the MATLAB (designated with "M") and Python (designated with "P") implementations.

i. Compute the SVD of the design matrix, $USV^T = X$ (M251, P151). To avoid numerical instability, very small singular values of X are treated as 0.
ii. The data are transformed $\tilde{y} = U^T y$ (M258, P62).
iii. The OLS problem is solved with one broadcast division operation (equation 10) (M276, P64).
iv. The values of α that correspond to the requested γ value are within a range that depends on the singular values of X (by equation 18). A series of initial candidate values of α are selected to span a log-spaced range from $10^{-3}\lambda_p^2$, much smaller than the smallest singular value of the design matrix, to $10^3\lambda_1^2$, much larger than the largest singular value of the design matrix (M302, P165-168). Based on testing on a variety of regression problems, we settled on a spacing of 0.2 log_{10} units within the range of candidate α values. This provides fine enough gridding such that interpolation results are nearly perfect (empirical fractions are approximately 1% or less from the desired fractions).
v. Based on equation 15, a scaling factor for every value of α and every singular value λ is calculated as (M316, P173):

$$SF_{i,j} = \lambda_i^2/(\lambda_i^2 + \alpha_j) \quad (23)$$

vi. The main loop of the algorithm iterates over targets. For every target, the scaling in equation 23 is applied to the computed OLS coefficients (from Step 3), and the L2-norm of the solution at each α_j is divided by the L2-norm of the OLS solution to determine the fractional length, γ_j (M336-349, P188-191). Because the relationship between α and γ may be different for each target, the algorithm requires looping over targets and cannot take advantage of broadcasting across targets.
vii. Interpolation is used with α_j and γ_j to find values of α that correspond to the desired values of γ (M367, P194). These target α values are then used to calculate the ridge regression solutions via equation 15 (M373, P203).
viii. After the iteration over targets is complete, the solutions are transformed to the original space by multiplying $\hat{\beta} = V\tilde{\hat{\beta}}$ (M422, P207).

In terms of performance, this algorithm requires just one (potentially computationally expensive) initial SVD of the design matrix. Operations done on a per-target basis are generally inexpensive, relying on fast vectorized array operations, with the exception of the interpolation step. Although a large range of candidate α values are evaluated internally by the algorithm, these values are eventually discarded, thereby avoiding costs associated with the final step (multiplication with V).

Software implementation

We implemented the algorithm described in section Algorithm in two different popular statistical computing languages: MATLAB and Python (example code in Figure 1). The code for both implementations is available at https://github.com/nrdg/fracridge and released under an OSI-approved, permissive open-source license to facilitate its broad use. In both MATLAB and Python, we used broadcasting to rapidly perform computations over multiple dimensions of arrays.

There are two potential performance bottlenecks in the code. One is the SVD step which is expensive both in terms of memory and computation time. In the case where $d < p$ (the number of data points is smaller than the number of parameters), the number of singular values is set by d. In the case where $d > p$ (the number of data points is larger than the number of parameters), the number of singular values is set by p, and our implementation exploits the fact that we can replace the singular values of X by the square roots of the singular values of $X^\intercal X$, which is only p by p. This optimization requires less memory for the SVD computation than an SVD of the full matrix X. The other potential performance bottleneck is the interpolation performed for each target. To optimize this step, we used fast interpolation functions that assume sorted inputs.

MATLAB

The MATLAB implementation of FRR relies only on core MATLAB functions and a fast implementation of linear interpolation [9], which is copied into the fracridge source code, together with its license, which is compatible with the fracridge license. The MATLAB implementation includes an option to automatically standardize predictors (either center or also scale the predictors) before regularization, if desired.

Python

The Python implementation of FRR depends on Scipy [10] and Numpy [11]. The object-oriented interface provided conforms with the API of the popular Scikit-Learn library [12, 13], including automated tests that verify compliance with this API (using Scikit Learn's `check_estimator` function, which automatically confirms this compliance). In addition to an estimator that fits FRR, a cross-validation object is implemented, using Scikit Learn's grid-search cross-validation API. Unit tests are implemented using pytest [14]. Documentation is automatically compiled using sphinx, with sphinx-gallery examples [15]. The Python implementation also optionally uses Numba [16] for just-in-time compilation of a few of the underlying numerical routines used in the implementation. This functionality relies on an implementation provided in the hyperlearn library [17] and copied into the fracridge source-code, together with its license, which is compatible with the fracridge license. In addition to its release on GitHub, the software is available to install through the Python Package Index (PyPI) through the standard Python Package Installer (`pip install fracridge`). For Python, we did not implement standardization procedures, as those are implemented as a part of Scikit-Learn.

Simulations

Numerical simulations were used to characterize FRR and compare it to a heuristic approach for hyperparameter selection. Simulations were conducted using the MATLAB implementation. We simulated two simple regression scenarios. The number of data points (d) was 100, and the number of predictors (p) was either 5 or 100. In each simulation, we first created a design matrix X (d, p) using the following procedure: (i) generate normally distributed values for X, (ii) induce correlation between predictors by selecting two predictors at random, setting one of the predictors to the sum of the two predictors plus normally distributed noise, and repeating this procedure $2p$ times, and (iii) z-scoring each predictor. Next, we created a set of "ground truth" coefficients β with dimensions $(p, 1)$ by drawing values from the normal distribution. Finally, we simulated

Matlab

```
y = randn(100,1);
X = randn(100,10);

% Calculate coefficients with naive OLS
coef = inv(X'*X)*X'*y;

% Call the fracridge function:
[coef2, alpha] = fracridge(X, 0.3, y);

% Calculate coefficients with naive RR
alphaI = alpha*eye(size(X, 2));
coef3 = inv(X'*X + alphaI)*X'*y;

norm(coef)
norm(coef2)
norm(coef2) ./ norm(coef)
norm(coef2-coef3)
```

Python

```
import numpy as np
from numpy.linalg import inv, norm
from fracridge import fracridge

y = np.random.randn(100)
X = np.random.randn(100, 10)

# Calculate coefficients with naive OLS
coef = inv(X.T @ X) @ X.T @ y

# Call fracridge function:
coef2, alpha = fracridge(X, y, 0.3)

# Calculate coefficients with naive RR
alphaI = alpha * np.eye(X.shape[1])
coef3 = inv(X.T @ X + alphaI) @ X.T @ y

print(norm(coef))
print(norm(coef2))
print(norm(coef2) / norm(coef))
print(norm(coef2 - coef3))

# sklearn-compatible object-oriented API:
from fracridge import FracRidgeRegressor
fr = FracRidgeRegressor(fracs=0.3)
fr.fit(X, y)
coef_oo = fr.coef_
alpha_oo = fr.alpha_
print(norm(coef_oo) / norm(coef))

# sklearn-style grid search cross-validation:
from fracridge import FracRidgeRegressorCV
frcv = FracRidgeRegressorCV(frac_grid=np.arange(0.1, 1, 0.1))
frcv.fit(X, y)
best_frac = frcv.best_frac_
print(best_frac)
print(norm(frcv.coef_) / norm(coef))
```

Figure 1. Code examples. Top: MATLAB examples that demonstrate the software API and correctness of the implementation. Bottom: Python examples demonstrate a similar API and correctness. Python examples include the Scikit-Learn-compatible API.

responses from the model ($y = X\beta$) and added normally distributed noise, producing a target variable y with dimensions $(d, 1)$.

Given design matrix X and target y, cross-validated regression was carried out. This was done by splitting X and y into two halves (50/50 training/testing split), solving ridge regression on one half (training) and evaluating generalization performance of the estimated regression β weights on the other half (testing). Performance was quantified using the coefficient of determination (R^2). For standard ridge regression, we evaluated a grid of α values that included 0 and ranged from 10^{-4} through $10^{5.5}$ in increments of 0.5 log_{10} units. For FRR, we evaluated a range of fractions γ from 0 to 1 in increments of 0.05. Thus, the number of hyperparameter values was $f = 21$ in both cases.

The code that implements these simulations is available in the "examples" folder of the software.

Performance benchmark

To characterize the performance of fractional ridge regression (FRR) and standard ridge regression (SRR) approaches, a set of numerical benchmarks was conducted using the MATLAB implementation. A range of regression scenarios were constructed. In each experiment, we first constructed a design matrix X (d, p) consisting of values drawn from a normal distribution. We then created "ground truth" coefficients β (p, t) also by drawing values from the normal distribution. Finally, we generated a set of data Y (d, t) by predicting the model response ($y = X\beta$) and adding zero-mean Gaussian noise with standard deviation equal to the standard deviation of the data from each target variable. Different levels of regularization (f) were obtained for SRR by linearly spacing α values on a log_{10} scale from 10^{-4} to 10^5 and for FRR by linearly spacing fractions from 0.05 to 1 in increments of 0.05.

Two versions of SRR were implemented and evaluated. The first version (naïve) involves a separate matrix pseudo-inversion for each hyperparameter setting desired. The second version (rotation-based) involves using the SVD decomposition method described above (see section Background and theory, specifically equation 14).

All simulations were run on an Intel Xeon E5-2683 2.10 Ghz (32-core) workstation with 128 GB of RAM, a 64-bit Linux operating system, and MATLAB 8.3 (R2014a). Execution time was logged for model fitting procedures only and did not include generation of the design matrix or the data. Likewise, memory requirements were recorded in terms of additional memory usage during the course of model fitting (i.e. zero memory usage corresponds to the total memory usage just prior to the start of model fitting). Benchmarking results were averaged across 15 independent simulations to reduce incidental variability.

The code that implements these benchmarks is available in the "examples" folder of the software.

Brain Magnetic Resonance Imaging data

Brain functional magnetic resonance imagining (fMRI) data were collected as part of the Natural Scenes Dataset (http://naturalscenesdataset.org). Data were acquired in a 7 Tesla MRI instrument, at a spatial resolution of 1.8 *mm* and a temporal resolution of 1.6 *s* and using a matrix size of [81 104 83]. This yielded a total of 783,432 voxels. Over the course of 40 separate scan sessions, a neurologically healthy participant viewed 10,000 distinct images (3 presentations per image) while fixating a small dot placed at the center of the images (see Figure 3A). The images were 8.4 deg by 8.4 deg in size. Each image was presented for 3 *s* and was followed by a 1 *s* gap. Standard pre-processing steps were applied to the fMRI data to remove artifacts due to head motion and other confounding factors. To deal with session-wise nonstationarities, response amplitudes of each voxel were z-scored within each scan session. Responses were then concatenated across sessions and averaged across trials of the same image, and then a final z-scoring of each voxel's responses was performed. The participant provided informed consent and the experimental protocol was approved by the University of Minnesota Institutional Review Board. For the purposes of the example demonstrated here, only the first 37 of the 40 scan sessions are provided (data are being held out for a prediction challenge), yielding a total of 9,841 distinct images.

A regression model was used to predict the response observed from a voxel in terms of local contrast present in the stimulus image. In the model, the stimulus image is preprocessed by taking the original color image (425 pixels by 425 pixels by 3 RGB channels), converting the image to grayscale, gridding the image into 25 by 25 regions, and then computing the standard deviation of luminance values within each grid region (Figure 4B). This produced 625 predictors, each of which was then z-scored. The design matrix X has dimensionality 9,841 images by 625 stimulus regions, while Y has dimensionality 9,841 images by 783,432 voxels.

Cross-validation was carried out using a 80/20 training/testing split. For standard ridge regression, we evaluated a grid of alpha values that included 0 and ranged from 10^{-4} to $10^{5.5}$ in increments of 0.5 log_{10} units. For fractional ridge regression, we evaluated a range of fractions from 0 to 1 in increments of 0.05. Cross-validation performance was quantified in terms of variance explained on the test set using the coefficient of determination (R^2).

The code that implements these analyses is available in the "examples" folder of the software.

Results

Fractional ridge regression achieves the desired outcomes

In simulations, we demonstrate that the fractional ridge regression (FRR) algorithm accurately produces the desired fractions γ (Figure 2 A,B second row, right column in each). We compare the results of FRR to results of standard ridge regression (SRR), in which a commonly-used heuristic is used to select α values (log-spaced values spanning a large range). For the SRR approach, we find that the fractional L2-norm is very small and virtually indistinguishable for large values of α, and is very similar to the OLS solution (fractional L2-norm approximately 1) for several small values of α (Figure 2 A, B second row, left column). In addition, cross-validation accuracy is indistinguishable for many of the values of α evaluated in SRR. Only very few values of α produce cross-validated R^2 values that are similar to the value provided by the best α (Figure 2 A, B first row, left column).

The SRR results can also be re-represented using effective degrees of freedom (DOF; Figure 2 A, B first row, middle column): several values of α result in essentially the same number of DOF, because these values are either much larger than the largest singular value or much smaller than the smallest singular value of X. In contrast to SRR, FRR produces a nicely behaved range of cross-validated R^2 values and dense sampling around the peak R^2.

Another line of evidence highlighting the diversity of the solutions provided by FRR is given by inspecting coefficient paths: in the log-spaced case, coefficients start very close to 0 (for high α) and rapidly increase (for lower α). Even when

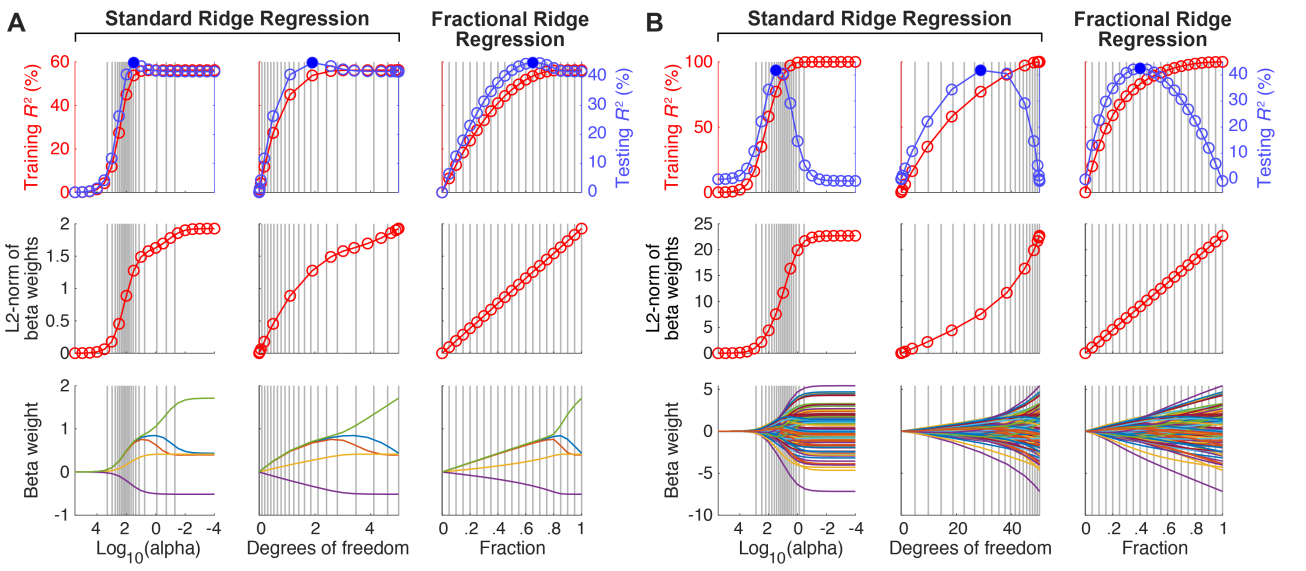

Figure 2. Fractional ridge regression (FRR) achieves desired outcomes. (A) Example regression scenario (d = 100, p = 5). The first two columns show the results of standard ridge regression in which log-spaced α values are used to obtain different levels of regularization. Whereas the first column shows results as a function of $log_{10}(\alpha)$, the second column shows results as a function of α values converted to effective degrees of freedom (see Methods). The third column shows the results of fractional ridge regression in which different regularization levels are achieved by requesting specific fractional L2-norm (γ). Solid blue dots mark peak cross-validation performance. Vertical gray lines in the third column indicate regression solutions obtained by the FRR method (requested fractions range from 0 to 1 in increments of 0.05). The corresponding locations of these regression solutions in the first and second columns are also shown using vertical gray lines. The bottom row shows coefficient paths, i.e., the values of β as a function of $log_{10}(\alpha)$, degrees of freedom, or fraction γ. (B) Example regression scenario (d = 100, p = 100). Same format as panel A. Notice that in both scenarios, only the FRR method achieves regression solutions whose L2-norms increase linearly, with gradually changing coefficient paths.

re-represented using DOF, the coefficient paths exhibit some redundancy. In contrast, FRR provides more gradual change in the coefficient paths, indicating that this approach explores the space of possible coefficient configurations more uniformly. Taken together, these analyses demonstrate that FRR provides a more useful range of regularization levels than SRR.

FRR is computationally efficient

A question of relevance to potential users of FRR is whether using the method incurs significant computational cost. We compare FRR to two alternative approaches. The first approach is a naïve implementation of the matrix inversion specified in equation 3, in which the Moore-Penrose pseudo-inverse (implemented as `pinv` in Matlab and `numpy.linalg.pinv` in Python) is performed independently for each setting of hyperparameter α. The second approach takes advantage of the computational expedience of the SVD-based approach: instead of a matrix inversion for each α value, a single SVD is performed, a transformation (rotation) is applied to the data, and different values of α are plugged into equation 14 to compute the regression coefficients. This approach comprises a subset of the operations taken in FRR. Therefore, it represents a lower bound in terms of computational requirements.

Through systematic exploration of different problem sizes, we find that FRR performs quite favorably. FRR differs from the rotation-based approach only slightly with respect to execution-time scaling in the number of data points (Figure 3A, left column), in the number of parameters (Figure 3A, right column), and in f, the number of hyperparameter values considered (Figure 3A, third column). The naïve matrix-inversion approach is faster than both SVD-based approaches (FRR and rotation-based) for $f < 20$, but rapidly becomes much more costly for values above 20. This approach also scales rather poorly for $p > 5,000$.

In terms of memory consumption, the mean and maximum memory usage are very similar for FRR and the naïve and rotation-based SRR solutions. These results suggest that for each of these approaches, the matrix inversion (for the naïve implementation of SRR) or the SVD (for FRR and the rotation-based SRR) represents the main computational bottleneck. Importantly, despite the fact that FRR uses additional gridding and interpolation steps, it does not perform substantially worse than either of the other approaches.

Application of FRR on real-world data

To demonstrate the practical utility of FRR, we explore its application in a specific scientific use-case. Data from a functional magnetic resonance imaging (fMRI) experiment were analyzed with FRR and the results of this analysis were compared to a standard ridge regression (SRR) approach where α values are selected using a log-spaced heuristic. Different parts of the brain process different types of information, and a large swath of the cerebral cortex is known to respond to visual stimulation. Experiments that combine fMRI with computational analysis provide detailed information about the responses of different parts of the brain [18]. In the experiments analyzed here, a series of images are shown and the blood-oxygenation-level-dependent (BOLD) signal is recorded in a sampling grid of voxels throughout the brain (Figure 4A). In the cerebral cortex, each voxel contains hundreds of thousands of neurons. If these neurons respond vigorously to the visual stimulus presented, the metabolic demand for oxygen in that part of cortex will drive a transient increase in oxygenated blood in that region, and the BOLD response will increase. Thus, a model of the BOLD response tells us about the selective responses of neurons in each voxel in cortex.

Because neurons in parts of the cerebral cortex that respond to visual stimuli are known to be particularly sensitive to local contrast, we model responses with respect to the standard deviation of luminance in each region of the image, rather than the luminance values themselves (Figure 4B). In the model, Y contains brain responses where each target (column) represents

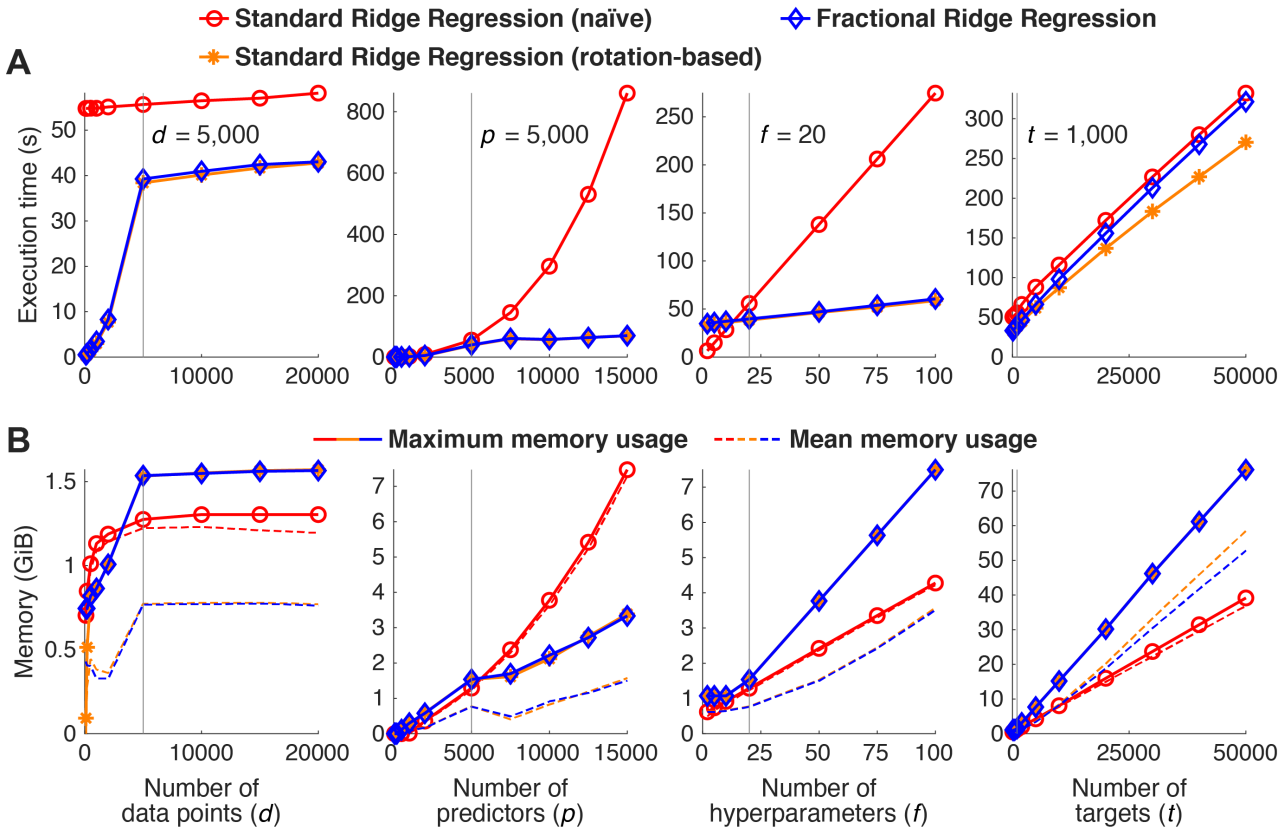

Figure 3. Computational efficiency. We benchmarked different methods for performing ridge regression: (1) a naïve implementation of standard ridge regression (involving log-spaced α values) in which matrix inversion is performed for each α value, (2) an implementation of standard ridge regression in which solutions are computed in a rotated space based on singular value decomposition of the design matrix, and (3) the FRR method. Starting from a base case ($d = 5,000$, $p = 5,000$, $f = 20$, $b = 1,000$; parameter settings marked by vertical lines), we systematically manipulated d, p, f, and b (columns one through four, respectively). (A) Execution time. The execution time of each method is shown in seconds. (B) Memory usage. The maximum memory usage of each method is shown as a solid line, whereas the time-averaged memory usage is shown as a dotted line. Overall, FRR is quite fast and has relatively modest memory requirements.

the responses in a single voxel. Each row contains the response of all voxels to a particular image. The design matrix X contains the local contrast in every region of the image, for every image. This means that the coefficients β represent weights on the stimulus image and indicate each voxel's spatial selectivity – i.e., the part of the image to which the voxel responds [19]. Therefore, one way to visualize $\hat{\beta}$ is to organize it according to the two-dimensional layout of the image (Figure 4C&D, bottom two rows).

Using FRR, we fit the model to voxel responses, and find robust model performance in the posterior part of the brain where visual cortex resides (left part of the horizontal slice presented in the top row of Figure 4C). The performance of the model can be observed in either the cross-validated R^2 values (Figure 4C, top row, left and middle panels) or the value of γ corresponding to the best cross-validated R^2 (top row, right panel). The γ values corresponding to best performance provide additional information about the differences between different targets, providing additional interpretation of the data. For example, we can focus on the two voxels highlighted in the middle panel of the top row in Figure 4C. One voxel, whose characteristics are further broken down in Figure 4D has lower cross-validated $R^2 = 4\%$ and requires stronger relative regularization ($\gamma = 0.15$). The spatial selectivity of this voxel's responses becomes very noisy at large γ values and R^2 approaches 0. On the other hand, the voxel in Figure 4E has a higher best $\gamma = 0.35$ and a higher cross-validated $R^2 = 13\%$. Moreover, this voxel appears more robust with higher values of γ producing less spatially noisy results. The map of R^2 and γ illustrated in Figure 4C also show that these trends hold more generally: voxels with more accurate models require less relative regularization. This demonstrates the additional interpretable information provided by the best γ values in individual targets and by inspecting spatial maps of these best γ values.

Discussion

The main theoretical contribution of this work is a novel approach to hyperparameter specification in ridge regression. Instead of the standard approach in which a heuristic range of values for hyperparameter α are evaluated for their accuracy, the fractional ridge regression (FRR) approach focuses on achieving specific fractions for the L2-norms of the solutions relative to the L2-norm of the unregularized solution. In a sense, this is exactly in line with the original spirit of ridge regression, which places a penalty on the L2-norm of the solution. The main practical contribution of this work is the design and implementation of an efficient algorithm to solve FRR and validation of this algorithm on simulated and empirical data. Note that the FRR algorithm can be viewed as method for finding appropriate α values that are adapted to the data such that they span the range of possible regularization strengths. Thus, it is fundamentally still a method that solves the standard ridge regression problem.

We emphasize that *in theory*, FRR and SRR are not expected to give different solutions to the linear regression problem. However, *in practice*, the solutions may very well differ and this will depend on the heuristic set of alpha values used in the SRR approach. What fractional ridge regression provides is a method to automatically ensure proper setting of alpha val-

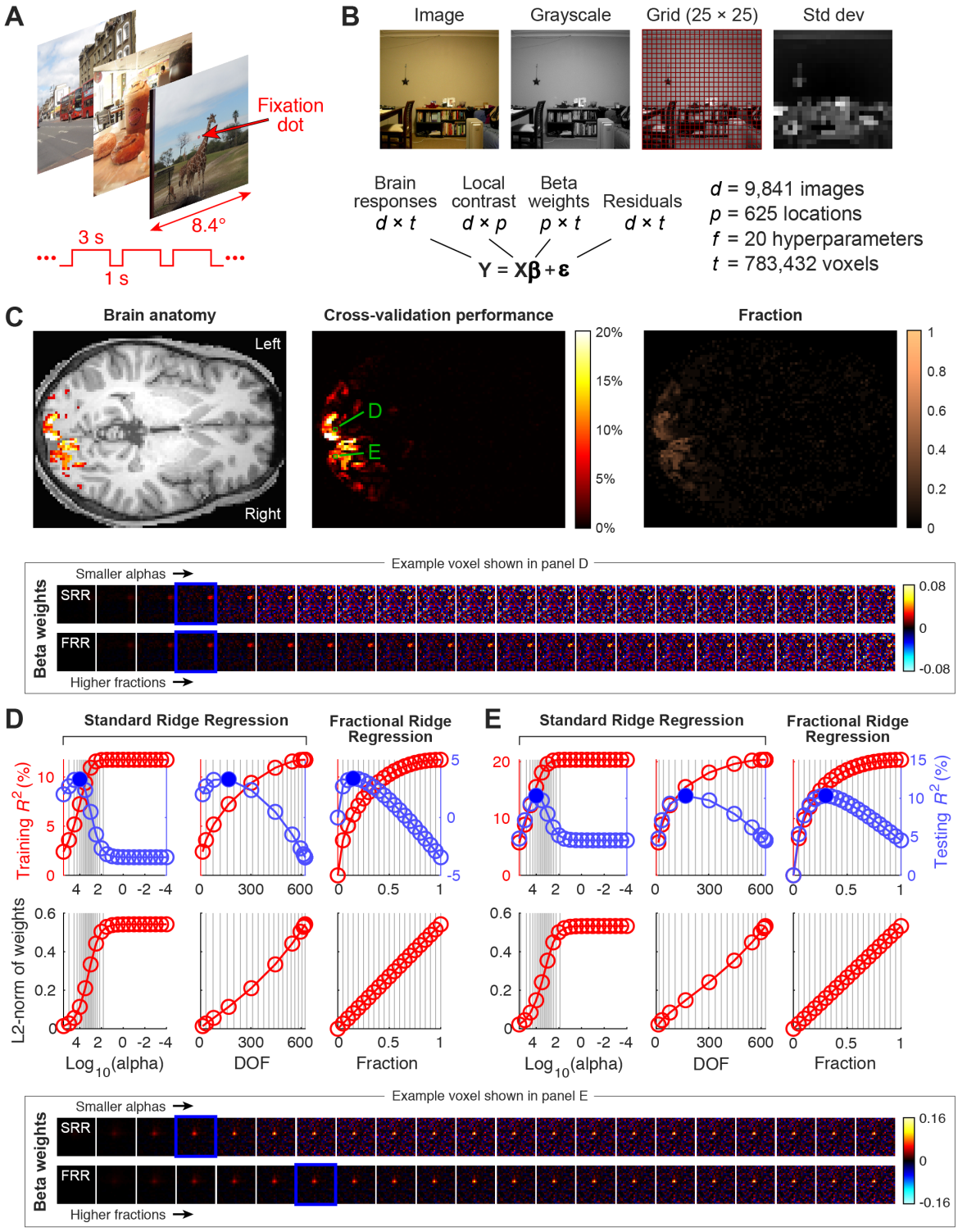

Figure 4. Demonstration on real-world data. (A) Visual fMRI experiment. Functional MRI measurements of brain activity were collected from a human participant while s/he viewed a series of natural images. (B) Model of brain activity. Images were converted to grayscale and gridded, and then standard deviation of luminance values within each grid element was calculated. This produced measures of local contrast. Brain responses at every voxel were modeled using a weighted sum of local contrast. (C) Results obtained using FRR. Cross-validated performance (variance explained) achieved by the model is shown for an axial brain slice (middle). These results are thresholded at 5% and superimposed on an image of brain anatomy for reference (left). The fraction (γ) corresponding to the best cross-validation performance is also shown (right). (D) Detailed results for one voxel (see green squares in panel C). The main plots that depict training and testing performance and L2-norm are in the same format as Figure 1. The inset illustrates coefficient solutions for different regularization levels. The blue box highlights the regularization level producing highest cross-validation performance. (E) Detailed results for a second voxel. Same format as panel D.

ues. Note that in the examples of SRR that we presented (e.g. Figure 2 and Figure 4), well-selected heuristic ranges of alpha values were used. This is done deliberately, as poor ranges of alpha values would have resulted in examples that are not very informative for this manuscript. However, in everyday practice, a user of the standard ridge regression approach might inadvertently use an inappropriate range of alpha values and obtain poor results. Overall, we suggest that FRR can serve as a default approach to solving ridge regression.

The benefits of FRR

i. **Theoretically-motivated and principled**. The results demonstrate that the theoretical motivation described in the Methods holds in practice. Our implementation of FRR produces ridge regression solutions that have predictable and tuneable fractional L2-norm.

ii. **Statistically efficient**. Each fraction level returned by FRR produces β values that are distinctly different. This avoids the common pitfall in the log-spaced approach whereby computation is wasted on several values of α that all over-regularize or under-regularize. When used with a range of γ values from 0 to 1, the solution that minimizes cross-validation error is guaranteed to exist within this range (although it may lie in between two of the obtained solutions).

iii. **Computationally efficient**. We show that our implementation of FRR requires memory and computational time that are comparable to a naïve ridge regression approach and to an approach that uses SVD but relies on preset α values. SVD-based approaches (including FRR) scale linearly in f, with compute-time scaling better than naïve RR in the $f > 20$ regime. In practice, we have found that $f = 20$ evenly distributed values between 0 and 1 provide sufficient coverage for many problems. But the linear scaling implies that sampling more finely would not be limiting in cases where additional precision is needed.

iv. **Interpretable**. FRR uses γ values that represent scaling relative to the L2-norm of the OLS solution. This allows FRR results to be compared across different targets within a dataset. This is exemplified in the results from an fMRI experiment that are interpreted both in light of cross-validated R^2 and the optimal γ that leads to the best cross-validated R^2. Moreover, regularization in different datasets and for different models (e.g., different settings of X) can be compared to each other as being stronger or weaker. The optimal regularization level can be informative regarding the signal-to-noise of a particular target or about the level of collinearity of the design matrix (which both influence the optimal level of regularization). FRR increases the interpretability of ridge regression, because instead of an unscaled, relatively inscrutable value of α, we receive a scaled, relatively interpretable value. Based on a recently proposed framework for interpretability in machine learning methods [20], we believe that this kind of advance improves the descriptive accuracy of ridge regression.

v. **Automatic**. Machine learning algorithms focus on automated inferences, but many machine learning algorithms still require substantial manual tuning. For example, if the range of α values used is not sufficient, users of ridge regression may be forced to explore other values. This is impractical in cases in which thousands of targets are analyzed and multiple models are evaluated. Thus, FRR contributes to the growing field of methods that aim to automate machine learning methods [21, 22]. These methods all aim to remove the burden of manual inspection and tuning of machine learning. A major benefit of FRR is therefore practical in nature: Because FRR spans the dynamic range of effects that ridge regression can provide, using FRR guarantees that the time taken to explore hyperparameter values is well spent. Moreover, the user does not have to spend time speculating what α values might be appropriate for a given problem (e.g. is 10^4 sufficiently high?).

vi. **Implemented in usable open-source software**. We provide code that is well-documented, thoroughly tested, and easy to use: https://github.com/nrdg/fracridge. The software is available in two popular statistical programming languages: MATLAB and Python. The Python implementation provides an object-oriented interface that complies with the popular Scikit-Learn library [12, 13].

Using FRR in practice

To select the level of regularization to apply in practice, users of FRR will likely use cross-validation. An open question is how to aggregate the results of FRR over multiple cross-validation splits. This is a general issue for any analysis that uses cross-validation to set hyperparameters. Nevertheless, here we provide some ideas for how users can apply FRR in practice: (i) one could determine the optimal fraction using cross-validation on a single training/testing split (e.g. 80/20), and obtain a single model solution and a corresponding optimal fraction, (ii) one could determine the optimal fraction using cross-validation on a single training/testing split and then adopt that fraction for solving the regression on the full dataset, with the understanding that this may yield a slightly over-regularized solution; (iii) one could determine the optimal fraction in different cross-validation splits of the data (e.g. n-fold cross-validation) and then average the determined fraction across the splits and average the estimated regression weights across the splits.

Fractional ridge regression is naturally integrated into a cross-validation framework where solutions reflecting different fractional lengths are obtained for a given set of data and evaluated for their predictive performance on held-out data. In the Python version of our software, this is implemented through an object that automatically performs a grid search to find the best value of γ among user-provided values. An alternative to performing cross-validation is the technique of generalized cross-validation (GCV). In GCV, for a given α value, matrix operations are used to efficiently estimate cross-validation performance without actually having to perform cross-validation [23]. It might be possible to combine the insights of FRR (e.g. the identification of interpretable and appropriate α values) with GCV.

Limitations

One limitation of FRR is that a heuristic approach is used within the algorithm to generate the grid of α values used for interpolation (see section for details). Nonetheless, the interpolation results are quite accurate, and costly computations are carried out only for final desired α values. Another limitation is that the α value that corresponds to a specific γ may be different for different targets and models. If there are theoretical reasons to retain the same α across targets and models, the FRR approach is not appropriate. But this would rarely be the case, as α values are usually not directly interpretable. Alternatively, FRR can be used to estimate values of α on one sample of the data (or for one model) and these values of α can then be used in all of the data (or all models).

Finally, the FRR approach is limited to ridge regression and does not generalize easily to other regularization approaches. The Lasso [24] provides regression solutions that balance least-squares minimization with the L1-norm of the coefficients, rather than the L2-norm of the coefficients. The Lasso approach has several benefits, including results that are more sparse and potentially easier to interpret. Similarly, Elastic Net [25] uses both L1- and L2-regularization, potentially offering more accurate solutions. But because the computational implementation of these approaches differs quite substantially from ridge regression, the approach presented in this paper does not translate easily to these methods. Moreover, while these methods allow regularization with a non-negativity constraint on the coefficients, this constraint is not easily incorporated into L2-regularization. On the other hand, a major challenge that

arises in L1-regularization is computational time: most algorithms operate for one target at a time and incur substantial computational costs, and scaling such algorithms to the thousands of targets in large-scale datasets may be difficult.

Future extensions

An important extension of the present work would be an implementation of these ideas in additional statistical programming languages, such as the R programming language, which is very popular for use in statistical analysis of data from many different domains. One of the most important tools for regularized regression is the glmnet software package which was originally implemented in the R programming language [26] and has implementations in MATLAB [27] and Python [28]. The software also provides tools for analysis and visualization of coefficient paths and of the effects of regularization on cross-validated error. The R glmnet vignette [29] demonstrates the use of these tools. In addition to identifying the α value that minimizes cross-validation error, glmnet also identifies the α which gives the most regularized model such that the cross-validated error is within one standard error of the minimum cross-validated error. This approach acknowledges that there is some error in selecting α and chooses to err on the side of a more parsimonious model [5]. Future extensions of FRR could implement this heuristic.

Acknowledgements

The authors would like to thank Noah Simon for helpful discussions and Noah Benson for comments on the manuscript.

Availability of source code and requirements

- Project name: Fractional Ridge Regression
- Project home page: http://github.com/nrdg/fracridge
- Operating system(s): Platform independent
- Programming language: Python and MATLAB
- License: 3-clause BSD
- Biotools URL: https://bio.tools/fracridge
- SciCrunch RRID: SCR_019045

Availability of supporting data and materials

Code and data to reproduce the figures in this manuscript are available under a CC-BY license through GigaDB [30].

Consent for publication

Consent to publish has been obtained from the fMRI subject as part of the informed consent procedure (see Methods).

Competing Interests

The authors declare no competing interests.

Funding

AR was funded through a grant from the Gordon & Betty Moore Foundation and the Alfred P. Sloan Foundation to the University of Washington eScience Institute, through NIH grants 1RF1MH121868-01 (PI: AR) from the National Institute for Mental Health and 5R01EB027585-02 (PI: Eleftherios Garyfallidis, Indiana University) from the National Institute for Biomedical Imaging and Bioengineering and through NSF grants 1934292 (PI: Magda Balazinska, University of Washington). KK was supported by NIH P41 EB015894. Collection of MRI data was supported by NSF IIS-1822683, NSF IIS-1822929, NIH S10 RR026783, and the W.M. Keck Foundation.

Author Contributions

AR and KK conceived the algorithm. AR and KK implemented software. KK conducted simulations and data analysis. AR and KK wrote the manuscript.

References

1. Hoerl AE, Kennard RW. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 1970;12(1):55–67.
2. Stone M. Cross-validation: A review. Statistics: A Journal of Theoretical and Applied Statistics 1978;9(1):127–139.
3. Stone M. Cross-validatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society: Series B (Methodological) 1974;36(2):111–133.
4. Tikhonov AN, Arsenin VY. Solutions of ill-posed problems. Wiley; 1977.
5. Friedman J, Hastie T, Tibshirani R. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software 2010;33(1):1.
6. Hastie T, Tibshirani R. Efficient quadratic regularization for expression arrays. Biostatistics 2004 Jul;5(3):329–340.
7. Skouras K, Goutis C, Bramson M. Estimation in linear models using gradient descent with early stopping. Statistics and Computing 1994;4(4):271–278.
8. Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning. Springer Series in Statistics, New York, NY, USA: Springer New York Inc.; 2001.
9. Mier JM, Quicker 1D linear interpolation: interp1qr; 2020. https://www.mathworks.com/matlabcentral/fileexchange/43325-quicker-1d-linear-interpolation-interp1qr.
10. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat Methods 2020 Mar;17(3):261–272.
11. van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science Engineering 2011 Mar;13(2):22–30.
12. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. the Journal of machine Learning research 2011;12:2825–2830.
13. Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, et al. API design for machine learning software: experiences from the scikit-learn project. arXiv preprint arXiv:13090238 2013;.
14. Krekel H, Oliveira B, Pfannschmidt R, Bruynooghe F, Laugher B, Bruhin F, pytest 5.4.1; 2004–. https://github.com/pytest-dev/pytest.
15. Òscar Nájera, Larson E, Estève L, Varoquaux G, Grobler J, Liu L, et al., sphinx-gallery/sphinx-gallery: Release v0.6.1. Zenodo; 2020. https://doi.org/10.5281/zenodo.3741781.
16. Lam SK, Pitrou A, Seibert S. Numba: a LLVM-based Python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC No. Article 7 in LLVM '15, New York, NY, USA: Association for Computing Machinery; 2015. p. 1–6.
17. Han-Chen D, hyperlearn; 2020. https://github.com/

821 danielhanchen/hyperlearn/.
822 18. Wandell B, Winawer J, Kay K. Computational Modeling of
823 Responses in Human Visual Cortex. In: Brain Mapping: An
824 Encyclopedic Reference Elsevier Inc.; 2015.p. 651–659.
825 19. Wandell BA, Winawer J. Computational neuroimaging
826 and population receptive fields. Trends Cogn Sci 2015
827 Jun;19(6):349–357.
828 20. Murdoch WJ, Singh C, Kumbier K, Abbasi-Asl R, Yu B.
829 Definitions, methods, and applications in interpretable
830 machine learning. Proc Natl Acad Sci U S A 2019
831 Oct;116(44):22071–22080.
832 21. Zöller MA, Huber MF. Benchmark and Survey of Auto-
833 mated Machine Learning Frameworks. arXiv 2019 Apr;.
834 22. Tuggener L, Amirian M, Rombach K, Lörwald S, Varlet A,
835 Westermann C, et al. Automated Machine Learning in Prac-
836 tice: State of the Art and Recent Results. arXiv 2019 Jul;.
837 23. Golub GH, Heath M, Wahba G. Generalized Cross-
838 Validation as a Method for Choosing a Good Ridge Param-
839 eter. Technometrics 1979 May;21(2):215–223.
840 24. Tibshirani R. Regression shrinkage and selection via the
841 lasso. Journal of the Royal Statistical Society Series B
842 (Methodological) 1996;p. 267–288.
843 25. Zou H, Hastie T. Regularization and variable selection via
844 the elastic net. Journal of the Royal Statistical Society: Se-
845 ries B (Statistical Methodology) 2005;67(2):301–320.
846 26. Friedman J, Hastie T, Tibshirani R. glmnet: Lasso and
847 elastic-net regularized generalized linear models. R pack-
848 age version 2009;1(4).
849 27. Qian J, Hastie T, Friedman J, Tibshirani R, Simon N,
850 Glmnet for matlab, 2013; 2013. http://www.stanford.edu/
851 hastie/glmnetmatlab.
852 28. Balakumar BJ, Hastie T, Friedman J, Tibshirani R, Simon
853 N, Glmnet for Python, 2016; 2016. https://web.stanford.
854 edu/~hastie/glmnet_python/.
855 29. Hastie T, Qian J, Glmnet vignette; 2014. http://www.web.
856 stanford.edu/~hastie/Papers/Glmnet_Vignette.pdf.
857 30. Kay K, Rokem A. Supporting data for "Fractional ridge re-
858 gression: a fast, interpretable reparameterization of ridge
859 regression". GigaScience Database 2020;http://dx.doi.
860 org/10.5524/100816.